Calibrating Artificial Neural Networks by Global Optimization

János D. Pintér[1,2]

1: Faculty of Engineering, Özyeğin University, Kusbakisi Caddesi, No.2, 34662 Istanbul, Turkey; www.ozyegin.edu.tr; janos.pinter@ozyegin.edu.tr.

2: Pintér Consulting Services, Inc., Canada; www.pinterconsulting.com; janos.d.pinter@gmail.com.

Technical Report, Özyeğin University, Istanbul

Submitted for publication: July 2010

ABSTRACT

An artificial neural network (ANN) is a computational model − implemented as a computer program − that is aimed at emulating the key features and operations of biological neural networks. ANNs are extensively used to model unknown or unspecified functional relationships between the input and output of a "black box" system. In order to apply such a generic procedure to actual decision problems, a key requirement is ANN training to minimize the discrepancy between modeled and measured system output. In this work, we consider ANN training as a (potentially) multi-modal optimization problem. To address this issue, we introduce a global optimization (GO) framework and corresponding GO software. The practical viability of the GO based approach is illustrated by finding close numerical approximations of (one-dimensional, but non-trivial) functions.

KEYWORDS

Artificial Neural Networks, ANN model calibration by global optimization, Lipschitz Global Optimizer (LGO) solver suite, ANN implementation in *Mathematica*, *MathOptimizer Professional* (LGO for *Mathematica*), Illustrative numerical examples.

1. Introduction

A neuron is a single nerve cell consisting of a nucleus (cell body, the cell's life support center), incoming dendrites with endings called synapses that receive stimuli (electrical input signals) from other nerve cells, and an axon (nerve fiber) with terminals that deliver output signals to other neurons, muscles or glands. The electrical signal traveling down on the axon is called neural impulse: this signal is communicated to the synapses of some other neurons. This way, a single neuron acts like a signal processing unit; the entire neural network has a complex, massive multi-processing architecture. Neural systems are essential for an organism, in order to adapt properly and quickly to its environment.

An artificial neural network (ANN) is a computational model − implemented as a computer program − that is aimed at emulating the essential features and operations of neural networks. ANNs are used extensively to model complex relationships between input and output data sequences, or to find hidden patterns in data sets when the analytical (explicit model function based) treatment of the problem is tedious or currently not possible. ANNs replace these unknown functional relationships by adaptively constructed approximating functions (approximators). Such approximators are typically

designed on the basis of training examples of a given task, such as recognising hand-written words. For in-depth expositions related to the ANN paradigm consult e.g. Hertz et al., (1991), Cichocki and Unbehauen (1993), Smith (1993), Golden (1996), Ripley (1996), Mehrotra et al. (1997), Bar-Yam (1999), Steeb (2005), Cruse (2006).

The mathematical model of an ANN is based on the key features of a neuron as outlined above. We shall consider first a single artificial neuron (AN), and assume that a finite sequence of input signals $s=(s_1, s_2, \ldots, s_T)$ is received by its synapses: the signals s_t $t=1,\ldots T$ may be real numbers or vectors. The AN's information processing capability will be modeled by introducing an aggregator (transfer) function a that depends on a certain parameter vector $x=(x_1, x_2, \ldots, x_n)$. Subsequently, x will be chosen according to a chosen optimality criterion. The optimization is typically carried out in such a way that the sequence of model output signals $m=(m_1, m_2, \ldots, m_T)$ generated by the function $a=a(x, s)$ is as close as possible to the corresponding set of observations $o=(o_1, o_2, \ldots, o_T)$.

This generic description can be summarized symbolically as follows.

$$s \rightarrow \text{AN} \rightarrow m \qquad s=(s_1, s_2, \ldots, s_T),\ m_t=a(x, s_t),\ t=1,\ldots T. \tag{1}$$

minimize $d(m, o)$ $m=(m_1, m_2, \ldots, m_T)$, $o=(o_1, o_2, \ldots, o_T)$, d is a discrepancy measure.

A single AN modeled as above is called a perceptron. Flexible generalizations of a single perceptron based model are multiple perceptrons receiving the same input signals: such a structure is called an artificial neuron layer (ANL). To extend this structure further, multilayered networks can also be defined: in these several ANLs are connected sequentially. It is also possible to model several functional relationships simultaneously by the corresponding output sequences of a multi-response ANN.

Figure 1 illustrates a multilayered feed-forward ANN that includes an input layer with multiple inputs, an intermediate (hidden) signal processing layer, and an output layer that emits multiple responses. The hidden layer's AN units receive the input information, followed by certain decomposition, extraction, and transformation steps to generate the output information. Hence, the ANN implementation versions used in practice all share the features of adaptive, distributed, local, and parallel processing.

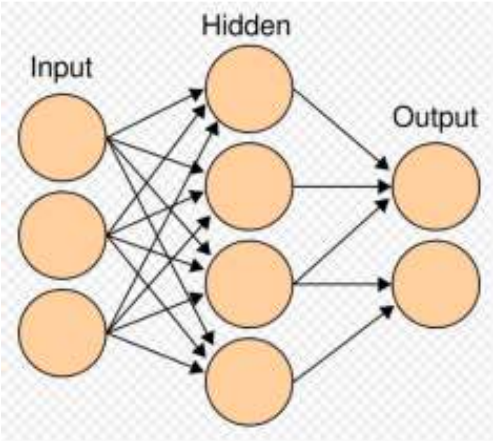

Figure 1 Schematic representation of a multilayered ANN
Source: http://en.wikipedia.org/wiki/Artificial_neural_network

Following the generic description (1), the optimal parameterization strategy is summarized as follows. Given the functional form of a and the sequence s of input values, we want to find a parameterization x such that the difference between the output set $m=a(x, s)$ and the observation set o is minimal. The discrepancy $d(m, o)$ is typically expressed by a suitable norm function: $d(m, o)=||m-o||$.

Once the parameter vector x becomes known − i.e. it has been estimated as a result of using the training I/O sequences s and o − the trained ANN can be used to perform various tasks related to the assessment of further input data such as $\{s_t\}$ for $t=T+1, T+2$, and so on. Some important application examples of this paradigm are signal processing, data classification, and time series forecasting; further application areas will be mentioned later on.

2. Function Approximation by ANNs: Theoretical Background and a Corresponding Optimization Model Formulation

The key theoretical feature of ANNs is their ability to approximate (unknown) continuous functions by learning from observed data, under general analytical conditions. Specifically, the approximation theorem of Cybenko (1989) states that a feed-forward ANN with a single hidden layer that contains a finite number of neurons equipped with a suitable transfer function can serve as a universal approximator of any given continuous real-valued function f: $\boldsymbol{R}^k \rightarrow \boldsymbol{R}$ defined on $[0,1]^k$. The approximation is uniform, in terms of the supremum norm of functions defined on $[0,1]^k$. We will discuss this result below in some detail, and then introduce a corresponding optimization model.

For a somewhat more complete picture of the theoretical foundations of ANNs, let us remark that Funahashi (1989) and Hornik et al. (1989) proved similar results to that of Cybenko. Hornik et al. (1989) proved that feed-forward neural networks with a single hidden layer of suitable transfer function units can approximate any Borel measurable multivariate vector function f:$\boldsymbol{R}^{k1} \rightarrow \boldsymbol{R}^{k2}$ uniformly, to arbitrary prescribed accuracy on an arbitrary compact set $C \subset \boldsymbol{R}^{k1}$. For more detailed technical discussions of these and closely related approximation results, consult also the foundations laid out by Kolmogorov (1957) which, however, do not lead directly to the results cited here since the approximating formula depends on the function f. More recent discussions of related function approximation results are presented by Park and Sandberg (1991, 1993), Lang (2005), Sutskever and Hinton (2008), with further topical references.

Let us consider a single input signal s_t from $\boldsymbol{R}^k$ $k \geq 1$, and the corresponding scalar output of $o_t = f(s_t)$ for all $t=1,..,T$: here f is an unknown continuous model function that we wish to approximate by a suitably defined ANN. We remark that in our general model the input signals can be delivered to a single (synapse of an) AN, or to several synapses of several ANs at the same time: the corresponding information will be then aggregated by the ANN.

Cybenko (1989) establishes the theoretical possibility of generating a parameterized approximating function $a(x, s_t)$, using a fixed type of transfer function σ such that the approximation shown below is uniformly valid.

$$f(s_t) \sim a(x, s_t) := \sum_{j=1,\ldots,J} \alpha_j \, \sigma(\sum_{i=1,\ldots,I} y_{ji} \, s_{ti} + \theta_j). \quad (2)$$

To relate this general result directly to an underlying ANN model, in (2) $j=1,\ldots,J$ denotes the index of the individual signal processing units AN_j in the hidden layer. For each unit AN_j, σ is a given function form to be specified below, α_j, y_j, and θ_j are model parameters to be determined depending on the unknown function f. Specifically, σ is a continuous real valued transfer function (in principle, σ could also be a function of j); $\sum_{i=1,\ldots,I} y_{ji} \, s_{ti}$ is the scalar product of the synaptic weight vector y_j of AN_j and the signal $s_t=(s_{t1},\ldots, s_{tI})$ component arriving from synapse i of AN_j, for $i=1,\ldots,I$; θ_j is called the threshold level (offset, bias) of AN_j; and α_j is the weight of AN_j in the function approximation (2) for $j=1,\ldots,J$.

Following Cybenko (1989) and Hornik et al. (1989), the transfer function σ is assumed to be monotonically non-decreasing, and it satisfies the relations

$$\lim \sigma(z)=0 \text{ as } z \to -\infty, \text{ and } \lim \sigma(z)=1 \text{ as } z \to \infty. \quad (3)$$

Transfer functions with these properties are called sigmoidal or squashing functions.

Let us remark here that other types of transfer functions can also be used in generating function approximations. Suitable sets of radial basis functions (RBF) are another well-known example (Park and Sandberg, 1991, 1993; Röpke et al., 2005). RBFs have the general form $\phi(z)=\phi(\|z-z_0\|)$: here $\|.\|$ is a given norm function, z_0 is the (scalar or vector) centre of the monotonically non-increasing function ϕ that satisfies the conditions $\phi(0)=1$, and $\lim \phi(z)=0$ as $\|z-z_0\| \to \infty$.

We will continue the present discussion using a specific sigmoidal transfer function form. An often used function form is

$$\sigma(z)=1/(1+e^{-z}) \qquad z \in \boldsymbol{R}. \quad (4)$$

For proper theoretical and better numerical tractability, we assume and define finite box (variable range) constraints to bound the values of the parameters α_j, y_{ji} and θ_j:

$$\begin{aligned} &\alpha_{jmin} \le \alpha_j \le \alpha_{jmax} && \text{for } j=1,\ldots,J. \\ &y_{jimin} \le y_{ji} \le y_{jimax} && \text{for } i=1,\ldots,I \text{ and } j=1,\ldots,J. \\ &\theta_{jmin} \le \theta_j \le \theta_{jmax} && \text{for } j=1,\ldots,J. \end{aligned} \quad (5)$$

The entire set of model parameters will be denoted by $x=(\{\alpha_j\}, \{y_{ji}\}, \{\theta_j\})$ where the appropriate components are included for all $i=1,\ldots,I$ and $j=1,\ldots,J$. The optimization problem induced for a given approximator function form is to minimize the error of the approximation (2) for the entire training sequence (s_t, o_t) $t=1,\ldots,T$, using a selected norm function. In this study, we shall use the least squares error criterion to measure the quality of the approximation: in other words, the objective function (6) shown below will be minimized under the box constraints (5).

$$e(x)=\sum_{t=1,\ldots,T} (f(s_t) - a(x, s_t))^2 \quad (6)$$

The notation $e(x)$ refers to the fact that (6) is an error function.

Let us point out here that other (more general linear or nonlinear) constraints regarding feasible parameter combinations may also be present if dictated by the problem context. Such situations can also be handled by our model development approach, by simply adding these constraints to the optimization model formulation. An important case in point is to attempt the exclusion of identical (symmetric) solutions to in which the component functions of the function a are in principle exchangeable. To break this symmetry, one can assume e.g. that the components of $\{\alpha_j\}$ are ordered so that

$$\alpha_j \geq \alpha_{j+1} \qquad \text{for all } j=1,\ldots,J\text{-}1. \tag{7}$$

One could also add further lexicographic constraints if dictated by the problem at hand. Another possible − but not always applicable − option could be to normalize the weights $\{\alpha_j\}$ so that we have

$$\sum_{j=1,\ldots,J} \alpha_j = 1, \quad \alpha_j \geq 0 \text{ for all } j=1,\ldots,J. \tag{8}$$

3. Postulating and Calibrating ANN Model Instances

Cybenko (1989) emphasizes that his cited result only guarantees a uniformly valid approximation of a continuous function f, as opposed to its exact representation. We should also keep in mind that the general approximation result expressed by (2) does *not* specify the actual number of processing units: hence, the key model parameter J needs to be estimated for each problem instance separately. Cybenko (1989) conjectures that J could become "astronomical" in many function approximation problems – especially so in higher dimensional approximations. These points should be kept in mind, to avoid making unjustifiably optimistic claims.

Once we select (test or postulate) the ANN structure to use in an actual case study, the remaining basic requirement is its application-specific parameterization with respect to a given training data set. The objective function in (6) makes this a nonlinear model fitting problem that could have a multitude of local or global optima. This multimodality issue is discussed in detail e.g. by Pintér (1996, 2003) with further references; in the context of ANNs, consult e.g. Auer et al. (1996), Bianchini and Gori (1996), Abraham (2004). For a more complete picture, let us remark that detailed expository discussions related to global optimization models, algorithms and applications are presented e.g. in the topical handbooks edited by Horst and Pardalos (1995), Pardalos and Romeijn (2002).

Let us also observe here that following the selection of the function form σ, the number of real-valued parameters (decision variables to optimize) is $J(I+2)$, each with corresponding box constraints; furthermore, the number of optionally added linear constraints − considering (7) and (8) − is J. Since in the general ANN model setting the values of I and J cannot be theoretically specified *a priori*, in practical applications one can try several computationally affordable combinations. In general, ANNs with more components could serve as better approximators, within reason and in line with the size of the training data set that should at least satisfy $T \geq J(I+2)$. At the same time, in order to assure the success of a suitable nonlinear optimization procedure (and a corresponding

software), one should restrict I and J to manageable values. This choice will then influence the accuracy of the approximation obtained.

Summarizing the preceding discussion, first one has to postulate an ANN structure, and the family of parameterized transfer functions to use. Next, one needs to provide a set of training (input-output) data. The training is then aimed at the selection of an optimally parameterized function from the chosen family of function models, to minimize the error function.

The iterative training of ANNs – based on local optimization techniques – is often referred to in the topical literature as back-propagation. Due to the possible multimodality of the error function (6), the simplest "standard" tools of unconstrained local nonlinear optimization (direct search, gradient based methods, Newton-type methods) often fail to locate the best parameter combination. This issue is well-known for ANN researchers: consult e.g. Bianchini and Gori (1996), Sexton et al. (1998), Abraham (2004), Hamm et al. (2007). To overcome this difficulty, the various optimization strategies used in practice to train ANNs include experimental design, sampling by low-discrepancy sequences, theoretically exact global or local scope search approaches, as well as popular heuristics such as evolutionary optimization methods, particle swarm optimization, simulated annealing, tabu search and tunneling functions. ANN model parameterization frameworks and numerical studies are presented and discussed e.g. by Watkin (1993), Prechelt (1994), Bianchini and Gori (1996), Sexton et al. (1998), Jordanov and Brown (1999), Toh (1999), Ye and Lin (2003), Abraham (2004), Hamm et al. (2007).

Next we will discuss a theoretically exact global optimization based approach and a corresponding software implementation that will be used subsequently to solve several illustrative ANN calibration problems.

Similarly to most other researchers, we will study "only" the problem of estimating the parameters of given ANNs with a given (postulated) architecture. The problem of finding the most appropriate model form in full generality seems elusive, and it is certainly difficult − if not impossible − to address in actual numerical studies.

4. Global Optimization: A General Modeling Framework

To match the modeling approach and notations introduced earlier, we shall use the following symbols and assumptions:

$x \in \boldsymbol{R}^n$ n-dimensional real-valued vector of decision variables

$e: \boldsymbol{R}^n \to \boldsymbol{R}$ continuous (scalar-valued) objective function

$D \subset \boldsymbol{R}^n$ non-empty set of feasible solutions, a proper subset of $\boldsymbol{R}^n$:

The feasible set D is closed and bounded, defined by

$l \in \boldsymbol{R}^n$, $u \in \boldsymbol{R}^n$ component-wise finite lower and upper bounds on x, and (optionally) by

$g: \boldsymbol{R}^n \to \boldsymbol{R}^m$ an m-vector of continuous (more general) constraint functions.

We shall then consider the continuous global optimization (GO) model

$$\min e(x) \quad \text{subject to } x \in D := \{x:\ l \le x \le u \quad g_j(x) \le 0 \quad j = 1,...,m\}. \tag{9}$$

Without going into details which are not necessary for the present discussion, let us remark that the terse model formulation (9) covers many special cases. Specifically, it also subsumes the ANN model calibration model statement summarized by formulas (4) to (6), with the optionally added constraints (7) and (8). Additionally, the GO problem statement (9) also guarantees that the set of global solutions is non-empty, and its structure supports the application of (theoretically) globally convergent deterministic and stochastic optimization algorithms. For technical details, consult e.g. Pintér (1996).

Needless to say, the numerical handling of GO model instances of (9) can be a tough challenge: even small-dimensional model instances can be hard to handle, and they essentially require an appropriate global scope search strategy. These general notes apply also to many ANN model instances as it will be illustrated shortly (in Section 6). Consequently, the calibration of ANNs – as a rule – requires proper GO software: one such software product will be briefly described next.

5. The LGO Software Package for Constrained Global-Local Optimization

Let us reiterate the key "black box" characteristic of the ANN modeling framework: namely, that its error function is evaluated by a numerical procedure that − in a model development and testing exercise − may be subject to changes and modifications. Similar situations that require (also) global scope search often arise in the real world of optimization: their solution requires easy-to-use, robust and efficient solver technology. Within the broad GO software category, direct (derivative-free) global search algorithm implementations have the advantage of immediate applicability to "black box" problems.

Next, we shall briefly review the key features of the Lipschitz Global Optimizer (LGO) solver suite that has been designed to directly address also "black box" problems. Developed since the late 1980's, LGO is currently available for a range of compiler platforms (C, C++, C#, Fortran 77/90/95), with seamless links to optimization modeling languages (AIMMS, AMPL, GAMS, MPL), MS Excel spreadsheets, and to the leading high-level technical computing systems Maple, Mathematica, and MATLAB. For theoretical, algorithmic and implementation details not discussed here, we refer e.g. to Pintér (1996, 1997, 2002, 2005, 2007, 2009, 2010a, 2010b), Pintér and Kampas (2005), Pintér et al. (2006).

The overall design of LGO is based on the combination of continuous nonlinear optimization strategies, with corresponding theoretical global and local convergence properties. Hence, LGO can be used for both constrained global and local optimization.

Let us point out here that LGO's overall derivative-free design is different from many other nonlinear optimization software packages that require explicit analytical model information to support model parsing and function type specific operations. Of course, we do not claim that one design is "always superior" to the other. A model function decomposition-parsing-bounding based approach supports the precise solution of a range of GO models, assuming sufficient runtime and computational resources. All models, however, have to be defined in terms of a given set of possible component functions. In contrast to this approach, the design of LGO allows the handling of genuine "black box" problems that will remain outside of the scope of the analytical global optimization approaches referred to above. For optimization model examples with embedded

computational procedures, consult e.g. Pintér (1996, 2009), Pintér et al. (2006), Kampas and Pintér (2006, 2009).

In numerical practice (that is in hardware resource-limited and time-limited runs), LGO's global search options generate a global solution estimate(s) that is (are) refined by the seamlessly following local search mode(s). This way, the expected result of using LGO is a global and local search based high-quality feasible solution that meets at least the local optimality criteria. (To guarantee theoretical local optimality, standard local smoothness conditions need to apply.) At the same time, one should keep in mind that no global – or, in fact, any other – optimization software will "always" work satisfactorily, with default settings and under resource limitations related to model size, time, model function evaluation, or to other preset usage limits.

Extensive numerical tests and an increasing range of widely different practical applications demonstrate that LGO and its platform-specific implementations can find high-quality numerical solutions not only when using academic GO test problems, but also in often far more complicated, sizeable GO models. Examples of real-world optimization challenges handled by various LGO implementations are environmental systems modeling and management (Pintér, 1996), laser design (Isenor et al., 2003), intensity modulated cancer therapy planning (Tervo et al., 2003), the operation of integrated oil and gas production systems (Mason et al., 2007), vehicle component design (Goossens et al., 2007), various packing problems with industrial relevance (Castillo et al., 2008), fuel processing technology development (Pantoleontos et al., 2009), and currency trading strategies (Çağlayan and Pintér, 2010).

6. Numerical Examples and Discussion

The detailed testing and comparative assessment of optimization software products is a significant research area. One needs to follow or to establish objective evaluation criteria, and to present (in principle) fully reproducible results. The software tests themselves are based on a given set of test problems and a selection of criteria such as the reliability and efficiency of the solvers used, in terms of the quality of solutions obtained and the corresponding computational effort. For examples of benchmarking studies, in the ANN context we refer to Prechelt (1994), Hamm et al. (2007); in the GO context, see e.g. Ali et al. (2005), Khompatraporn et al. (2005), Pintér (2002, 2007); with further topical references therein.

Here we will illustrate the performance of global optimization software (specifically, of an LGO implementation) in the ANN calibration context, by solving merely one-dimensional, but non-trivial function approximation problems. The key features of the computational environment used and the numerical tests are summarized below: the complete set of detailed numerical results is available from the author upon request.

- Hardware: PC, Intel™Core™2 Duo CPU P8400 @ 2.26GHz, 2.86 GB RAM.
- Operating System: Windows XP Pro 2002 with SP 3 (32-bit).
- Global optimization software: *MathOptimizer Professional* (MOP). Let us remark that *MathOptimizer Professional* is the software product name introduced for the LGO solver implementation that is linked to *Mathematica* (Wolfram, 2009). For details, consult e.g. Pintér and Kampas (2005), Kampas and Pintér (2006).

- ANN computational models used: 1-*J*-1 type networks that have a single input node, a single output node, and *J* processing units. Recall that this setup directly follows the function approximation structure expressed by (2). The ANN models considered are implemented in native *Mathematica* code.
- ANN initialization by assigned starting parameter values is not necessary, since MOP (i.e., LGO) uses a genuine global scope optimization approach. Let us note at the same time that all LGO implementations also support the optional use of set initial values in their constrained local optimization mode, without applying first a global search mode.
- A single run is done in all cases; no repetitions are needed, since MOP produces identical runs under the same initializations and solver options. Repeated runs with possibly changing numerical results can be easily generated by setting several options of MOP. The latter are based on selecting one of the several global search operational modes, using different starting points in the local search mode, using different random seeds, and assigning different computational resources to MOP runs.
- In all test problems the optimized function approximations are based on a set of training points, in line with the modeling framework of Section 2.
- Simulated noise structure: none. In other words, we assume that all training data are exact. This assumption could be easily relaxed, by introducing a probabilistic noise structure as done e.g. in Pintér (2003).
- Key performance measure: standard error (the root of the mean square error, RMSE) calculated for the numerical solution found.
- Some additional *Mathematica* features used: documentation and model visualization capabilities.
- Some additional MOP features used here (or elsewhere): automatic generation of optimization model code in C or Fortran; and the automatic generation of input and result text files. These files are used and generated by the LGO solver core of MOP.

In the numerical examples presented here, our objective is to "learn" the graph of certain one-dimensional functions defined on the interval [0,1] based on a preset number of uniformly spaced training points. As earlier, we shall use the notation *T* for the number of training data, and *J* for the number of processing units in the hidden layer.

Example 1

Our first (easiest) test problem presented here has been studied also by Toh (1999), as well as by Ye and Lin (2003).

$$f(x)=sin(2\pi x)\ cos(4\pi x) \qquad 0\le x\le 1. \tag{10}$$

Applying T=100, J=7, and MOP's local search mode (LS), we obtain a corresponding function approximation characterized by RMSE ~ 0.00294674. With default MOP settings, the number of error function evaluations is 83,473; the corresponding runtime is 4.20 seconds, implying an estimated 20,000 function evaluations per second. (Let us remark that the speed estimate can vary a bit, based on the actual status of the computer used.) For comparison, the function (10) is displayed below (see the left hand side of

Figure 2), directly followed by list plot (right hand side) produced by the output of the trained ANN at the uniformly distributed T=100 sample points.

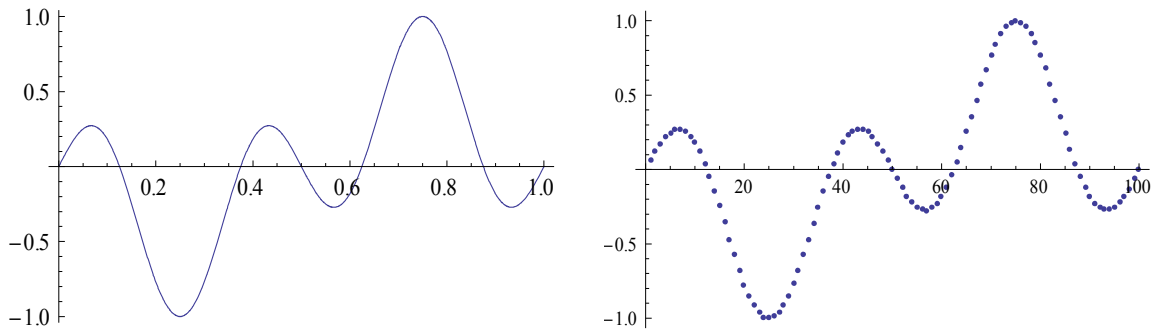

Figure 2 Function (10), and the list plot produced by the optimized ANN.

Both the numerical result (the RMSE obtained) and Figure 2 indicate a good fit. For a more complete picture, we also solved the same problem using LGO's most extensive global search mode (stochastic multi-start, MS) followed by corresponding local searches from the best points found. This operational mode is referred to as MSLS, and it has been used in all subsequent numerical examples.

In the MSLS mode, we assume (postulate) the variable ranges [*lb*, *ub*] = [-20, 20] for all ANN model parameters, and set a limit for the global search steps (error function evaluations) as *gss*=10^6. Based on our GO related numerical experience, this does not seem to be an exorbitant setting since we have to optimize 21 parameters globally. (For a simple comparison, one can think of aiming at just 10% precision for each variable setting in 21 dimensions, using a uniform grid based search: this naïve strategy would require 10^{21} function evaluations.)

As a result of using the MSLS operational mode, we receive RMSE ~ 0.00166644 which represents about 43.5% reduction of the average error found using only LS. The corresponding runtime is 59.77 seconds.

It is also worth pointing out that the optimized ANN parameterizations found by LS and MSLS are quite different. Several components of these solutions lie on the boundary of the preset interval range [-20, 20] indicating a possible need for extending the search region. These findings, together with our other runs for this and other test problems indicate the potentially massive multi-modality of the objective (4) in similar or more complicated function approximation problems as well as the possible need to search on larger parameter regions.

In this particular test example, we could have perhaps reduced the global search effort, or just use the local search based estimate which seems "good enough". Such experimental "streamlining" may not always work, however, and we are fully convinced that global scope search is typically needed in calibrating ANNs. Hence, we will continue to demonstrate MOP's standard global optimization capabilities, by solving all further test problems in MSLS mode.

The examples presented below are chosen to be increasingly more difficult, although all are "only" approximation problems related to one-dimensional functions. Obviously, one could "fabricate" functions that become very difficult to match by ANNs.

Example 2

$$f(x)=sin(2\pi x)\ sin(3\pi x)\ sin(5\pi x) \quad 0\leq x\leq 1. \tag{11}$$

To handle this problem, we use T=100, J=10, and MOP's MSLS search mode. Notice that the number of processing units has been increased heuristically, reflecting our belief that this function could be more difficult to reproduce than (10). This leads to a 30-variable GO model: we keep the allocated global search effort the same (gss=10^6) as in Example 1, however.

We conjecture again that the induced GO model is highly multi-extremal. The global search based numerical solution has an RMSE ~ 0.00493399; the corresponding runtime is 76.16 seconds (1,053,014 function evaluations, i.e. ~ 13,800 evaluations per second). For comparison, the function (11) is displayed below, directly followed by the list plot produced by the trained ANN, see Figure 3. Again, the reproduction of the target function seems to be quite satisfactory, both numerically and visually.

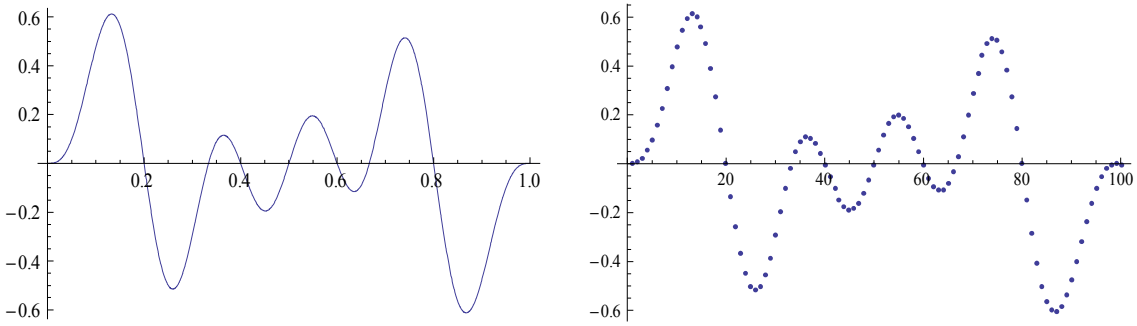

Figure 3 Function (11), and the list plot produced by the optimized ANN.

Example 3

$$f(x)=sin(5\pi x)\ sin(7\pi x)\ sin(11\pi x)\ log(1+x) \qquad 0\leq x\leq 1. \tag{12}$$

In (12), *log* denotes the natural logarithm function.

As above, first we use T=100, J=10, and MOP's MSLS search mode with 10^6 global search steps; furthermore, we extend the search region to [-50, 50] for each parameter. The global search based numerical solution has an RMSE ~ 0.0353158 which is noticeably not as good as the previous RMSEs; the corresponding runtime is 82.50 seconds. For comparison, the function (12) is displayed below, directly followed by the list plot produced by the trained ANN. In the corresponding Figure 4, one can notice a more apparent discrepancy in the "low-amplitude" section of the function, estimated by simple inspection as the interval [0, 0.4], while the approximation is quite satisfactory in the remaining "higher-amplitude" interval [0.4, 1].

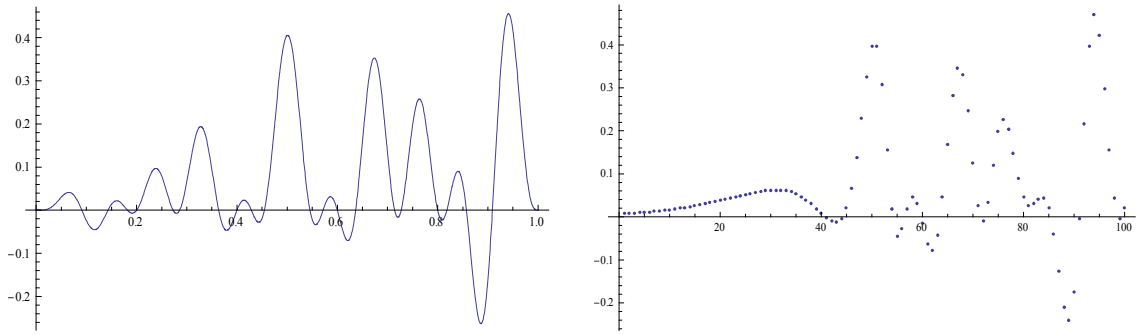

Figure 4 Function (12), and the list plot produced by a suboptimal ANN.

For comparison, we also ran MOP in its (only) LS mode; the resulting standard error is RMSE ~ 0.0617162; the runtime is 28.19 seconds. Figure 5, corresponding to this case, visibly becomes a much more crude approximation of (12) than the list plot in Figure 4. Again, this finding illustrates the typical need for a global scope model fitting approach.

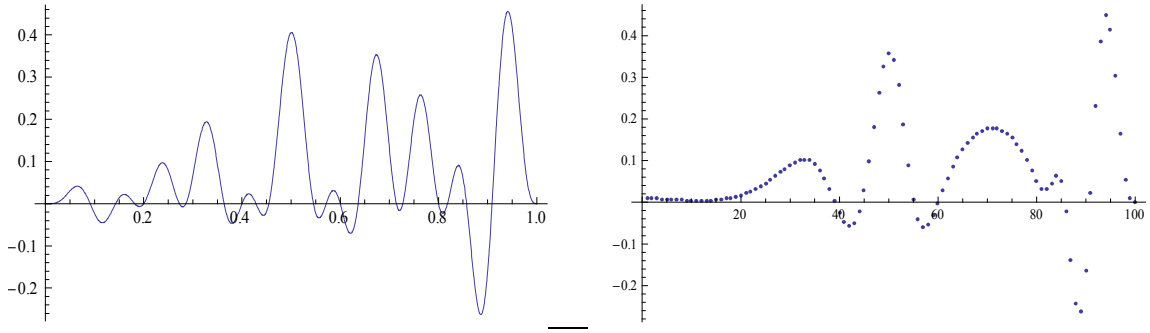

Figure 5 Function (12), and the list plot produced by the locally optimized ANN.

Example 4

In several subsequent runs, we increased the required precision of the approximation to function (12) by the ANN, by increasing one or several of the following: the number of training data *T*, the number of processing units *J*, and the solver runtime allocated to MOP. Obviously, all the above measures could lead to improved results.

Without going into unnecessary further details, Figure 6 (as an example) illustrates that e.g. the combination *T*=200, *J*=20, *gss*=3,000,000, and the additionally set runtime limit of 900 seconds allows the approximation of the entire function rather faithfully. This finding is also indicated by the corresponding RMSE ~ 0.0111696. Let us remark that this MOP run was automatically interrupted at the set time limit: arguably, without this limit even better approximations could be found.

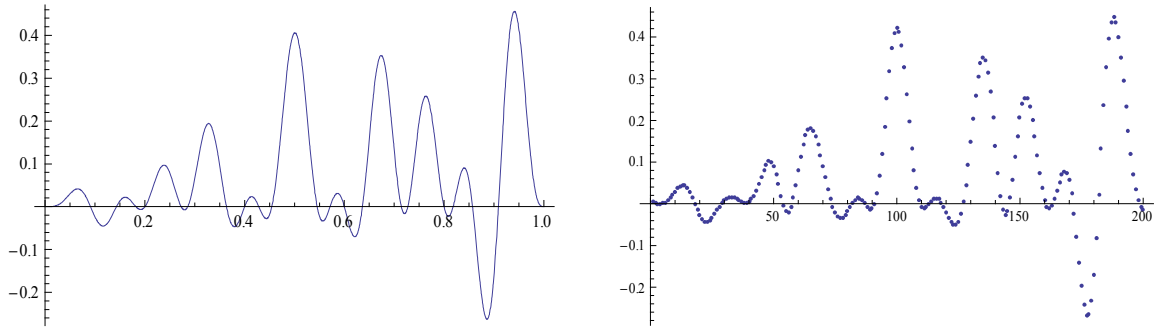

Figure 6 Function (12), and the list plot produced by the globally optimized ANN.

To summarize our illustrative numerical experiments, we can say that the GO based ANN calibration approach can be used to handle non-trivial function approximation problems within reasonable computational effort (on today's personal computers). Let us recall that – in light of the theoretical results cited earlier – exact function representations are in general too much to ask for.

Our approach is directly applicable to solve at least certain types of multi-input and multi-response approximation problems. If the input sequences need to be aggregated then this will lead to increase GO model dimensionality. However, if the input data sequences can be processed by independently operated hidden layers for each function output, then our approach can be applied iteratively (repetitively) to such multi-response problems. Hence, in case of fully decoupled ANN models, the corresponding sub-problems can be solved independently, thereby avoiding the (expected) exponential increase of computational complexity. Notwithstanding the above, one should be careful to avoid overly optimistic blanket statements when handling general multiple-input multiple-output problems by ANNs.

6. Conclusions

Due to its "universal approximator" capability, the ANN computational structure flexibly supports the modeling of many important research problems and real world applications that can be described by continuous functions defined over compact sets. ANN application areas includes data classification and pattern recognition (Ripley, 1996), damage detection and earthquake simulation (Pei et al., 2006), function approximation (Toh, 1999; Ye and Lin, 2003), material science (Bhadeshia, 1999), experimental design of engineering systems (Röpke et al., 2005), nonlinear optimization (Malek et al., 2010), polypeptide structure prediction (Dorn and de Souza, 2010), prediction of trading signals of stock market indices (Tilakaratne et al., 2008), regression analysis (De Veux et al., 1998), signal and image processing (Watkin, 1993; Masters, 1994), time series analysis and forecasting (Franses and van Dijk, 2000; Kajitani et al., 2005).

At the same time, the ANN computational model itself has an obvious meta-heuristic flavor in the sense that it needs to be instantiated and calibrated for each new model type. Even if such problem-specific ANN design can benefit from the expertise of domain specialists, manual design can become truly difficult when problem complexity increases.

Furthermore, once the ANN structure is selected, the computational effort may still need adjustments to handle the application satisfactorily. These practical aspects require flexible and transparent ANN implementations. Our current research code set up in an interpreted computing environment (*Mathematica*) is an illustrative example of such an implementation. After defining the function to be approximated, one needs to change only a few input data to produce a new sampling plan, to define the ANN instance (within a postulated ANN family), and to activate the optimization engine. The corresponding results can be then directly observed and visualized (after a few seconds or minutes in the examples presented here, using an average capability PC).

A focal point of the present study has been to demonstrate the essential requirement of high-quality ANN model calibration. To meet this challenge, we apply a global optimization (GO) based model fitting strategy. Our approach is based on theoretically sound GO methodology, and the corresponding software implementations are capable of handling non-trivial function approximation problems in reasonable time. The presented illustrative examples directly indicate the applicability of the GO based approach e.g. to pattern or symbol recognition, image or signal processing, and time series analysis.

In accordance with the theoretically exponential complexity of GO, the numerical demand of model fitting exercises can be expected to increase considerably with the size of the induced optimization model. This specifically includes ANN fitting problems in higher dimensions, and the consideration of general (possibly complicated) constraints regarding feasible parameter settings. We plan to investigate these issues in forthcoming studies, in relation to practically motivated problems that can be modeled by ANNs.

Acknowledgements

The author wishes to acknowledge the valuable contributions Dr. Frank Kampas to the development of the *MathOptimizer Professional* software product. Thanks are due also to Dr. Maria Battarra for her comments on the manuscript.

References

Abraham, A. (2004) Meta learning evolutionary artificial neural networks. *Neurocomputing* 56, 1-38.

Ali, M.M., Khompatraporn, C. and Zabinsky, Z.B. (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization* 31, 635–672.

Auer, P. Herbster, M. Warmuth, M. (1996) Exponentially many local minima for single neurons. In: Touretzky, D. et al., (Eds.), *Advances in Neural Information Processing Systems, Vol. 8*, pp. 316–322. MIT Press, Cambridge, MA.

Bar-Yam, Y. (1999). *Dynamics of Complex Systems*. Westview Press, Boulder, CO.

Bhadeshia, H.K.D.H. (1999) Neural networks in materials science, *ISIJ International*, 39 (10), 966-979.

Bianchini, M. and Gori, M. (1996). Optimal learning in artificial neural networks: A review of theoretical results. *Neurocomputing* 13, 313-346.

Çağlayan, M.O. and Pintér, J.D. (2010) Development and calibration of currency market strategies by global optimization. Research Report, Özyeğin University, Istanbul. (Submitted for publication)

Castillo, I., Kampas, F.J., and Pintér, J.D. (2008) Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research* 191, 786–802.

Cichocki, A. and Unbehauen, R. (1993) *Neural Networks for Optimization and Signal Processing*. John Wiley & Sons, Chichester, UK.

Cruse, H. (2006) *Neural Networks as Cybernetic Systems* (2nd and Revised Edition). Brains, Minds and Media, Bielefeld, Germany.

Cybenko, G. (1989) Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems* 2, 303-314.

De Veux, R.D., Schumi, J., Schweinsberg, J., and Ungar, L.H. (1998) Prediction intervals for neural networks via nonlinear regression. *Technometrics* 40, 273–282.

Dorn, M., and de Souza, O. N. (2010) A3N: An artificial neural network N-gram-based method to approximate 3-D polypeptides structure prediction. *Expert Systems with Applications* (forthcoming), doi:10.1016/j.eswa.2010.04.096.

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

Funahashi, K. (1989) On the approximate realization of continuous mappings by neural networks. *Neural Networks* 2, 183-192.

Golden, R.M. (1996) *Mathematical Methods for Neural Network Analysis and Design*, MIT Press, Cambridge, MA.

Goossens, P., McPhee, J., Schmitke, C., Pintér, J.D., and Stahl, H. (2007) Driving innovation: how mathematical modeling and optimization increase efficiency and productivity in vehicle design. *Technical Memorandum*, Maplesoft, Waterloo, ON.

Hamm, L., Brorsen, B.W. and Hagan, M. T. (2007) Comparison of stochastic global optimization methods to estimate neural network weights. *Neural Processing Letters* 26, 145–158.

Hertz, J., Krogh, A., and Palmer, R.G. (1991) *Introduction to the Theory of Neural Computation.* Addison-Wesley, Reading, MA.

Hornik, K., Stinchcombe, M., and White, H. (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359–366.

Horst, R. and Pardalos, P.M., Eds. (1995) *Handbook of Global Optimization*, Volume 1. Kluwer Academic Publishers, Dordrecht.

Isenor, G., Pintér, J.D. and Cada, M. (2003) A global optimization approach to laser design. *Optimization and Engineering* 4, 177-196.

Jordanov, I. and Brown, R. (1999) Neural network learning using low-discrepancy sequence. In Foo, N., Ed., *AI'99*, *Lecture Notes in Artificial Intelligence* 1747, pp. 255-267. Springer-Verlag, Berlin Heidelberg.

Kajitani, Y., Hipel, K.W., and McLeod, A.I. (2005) Forecasting nonlinear time series with feed-forward neural networks: a case study of Canadian lynx data. *Journal of Forecasting* 24, 105–117.

Kampas, F.J. and Pintér, J.D. (2006) Configuration analysis and design by using optimization tools in *Mathematica*. *The Mathematica Journal* 10 (1), 128-154.

Kampas, F.J. and Pintér, J.D. (2009) *MathOptimizer*: A nonlinear optimization package for *Mathematica* users. Research Report, Özyeğin University, Istanbul. (Submitted for publication)

Khompatraporn, Ch., Pintér, J.D. and Zabinsky, Z.B. (2005) Comparative assessment of algorithms and software for global optimization. *Journal of Global Optimization*, 31, 613–633.

Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSR*, 114, 953-956.

Lang, B. (2005) Monotonic multi-layer perceptron networks as universal approximators. In: Duch, W. et al., Eds., *Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005*, pp. 31-37. *Lecture Notes in Computer Science 3697*, Springer-Verlag Berlin Heidelberg.

Malek, A., Hosseinipour-Mahania, N., and Ezazipoura, S. (2010) Efficient recurrent neural network model for the solution of general nonlinear optimization problems. *Optimization Methods & Software* 25 (4), 489–506.

Mason, T.L., Emelle, C., van Berkel, J., Bagirov A.M., Kampas, F. and Pintér, J.D. (2007) Integrated production system optimization using the Lipschitz Global Optimizer and the Discrete Gradient Method. *Journal of Industrial and Management Optimization* 3 (2), 257-277.

Masters, T. (1994) *Signal and Image Processing with Neural Networks*, John Wiley & Sons, New York.

Mehrotra, K., Mohan, C.K., and Ranka, S. (1997) *Elements of Artificial Neural Nets.* MIT Press, Cambridge, MA.

Pantoleontos, G., Basinas, P., Skodras, G., Grammelis, P., Pintér, J.D., Topis, S., and Sakellaropoulos, G.P. (2009) A global optimization study on the devolatilisation kinetics of coal, biomass and waste fuels. *Fuel Processing Technology* 90, 762–769.

Pardalos, P.M. and Romeijn, H.E., Eds. (2002) *Handbook of Global Optimization*, Volume 2. Kluwer Academic Publishers, Dordrecht.

Park, J. and Sandberg, I.W. (1991) Universal approximation using radial-basis-function networks. *Neural Computation* 3 (2), 246-257.

Park, J. and Sandberg, I.W. (1993) Approximation and radial-basis-function networks. *Neural Computation* 5 (2), 305-316.

Pei, J-S., Mai, E., and Piyawat, K. (2006) Multilayer feedforward neural network initialization methodology for modeling nonlinear restoring forces and beyond. 4WCSCM-306, *4th World Conference on Structural Control and Monitoring*, San Diego, CA, July 2006.

Pintér, J.D. (1996) *Global Optimization in Action*. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (1997) LGO – A program system for continuous and Lipschitz optimization. In: Bomze, I.M., Csendes, T., Horst, R. and Pardalos, P.M., Eds. *Developments in Global Optimization*, pp. 183-197. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (2002) Global optimization: software, test problems, and applications. In: Pardalos, P.M. and Romeijn, H.E., Eds. *Handbook of Global Optimization*, Volume 2, pp. 515-569. Kluwer Academic Publishers, Dordrecht.

Pintér, J.D. (2003) Globally optimized calibration of nonlinear models: techniques, software, and applications. *Optimization Methods and Software* 18 (3) 335-355.

Pintér, J.D. (2005) Nonlinear optimization in modeling environments: software implementations for compilers, spreadsheets, modeling languages, and integrated computing systems. In: Jeyakumar, V. and Rubinov, A.M., Eds., *Continuous Optimization: Current Trends and Modern Applications,* pp. 147-173. Springer Science + Business Media, New York.

Pintér, J.D. (2007) Nonlinear optimization with GAMS/LGO. *Journal of Global Optimization* 38, 79-101.

Pintér, J.D. (2009) Software development for global optimization. In: Pardalos, P.M. and T. F. Coleman, eds. Global Optimization: Methods and Applications, pp. 183-204. *Fields Institute Communications Volume 55*. Published by the American Mathematical Society, Providence, RI.

Pintér, J.D. (2010a) *LGO ─ A Model Development and Solver System for Nonlinear (Global and Local) Optimization. User's Guide.* (Current version) Distributed by Pintér Consulting Services, Inc., Canada; www.pinterconsulting.com.

Pintér, J.D. (2010b) *Spreadsheet-Based Modeling and Nonlinear Optimization with Excel-LGO. User's Guide.* With technical contributions by B.C. Şal and F.J. Kampas. (Current version) Distributed by Pintér Consulting Services, Inc., Canada. www.pinterconsulting.com.

Pintér, J.D. and Kampas, F.J. (2005) Nonlinear optimization in *Mathematica* with *MathOptimizer Professional*. *Mathematica in Education and Research* 10 (2), 1-18.

Pintér, J.D., Linder, D., and Chin, P. (2006) Global Optimization Toolbox for Maple: an introduction with illustrative applications. *Optimization Methods and Software* 21 (4), 565-582.

Prechelt, L. (1994) PROBEN1 − A set of benchmarks and benchmarking rules for neural network training algorithms. *Technical Report 21/94*, Fakultat fur Informatik, Universitat Karlsruhe, Germany, September 1994.

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, MA.

Röpke, K. et al., Eds. (2005) *DoE – Design of Experiments*. Produced with the technical collaboration of IAV GmbH. Verlag Moderne Industrie. © sv corporate media GmbH, Munich, Germany.

Sexton, R.S., Alidaee, B., Dorsey, R.E. and Johnson, J.D. (1998) Global optimization for artificial neural networks: A tabu search application. *European Journal of Operational Research* 106 (1998) 570-584.

Smith, M. (1993) *Neural Networks for Statistical Modeling*, Van Nostrand Reinhold, New York.

Steeb, W-H. (2005) *The Nonlinear Workbook*. (3rd Edition) World Scientific, Singapore.

Sutskever, I. and Hinton, G.E. (2008) Deep, narrow sigmoid belief networks are universal approximators. *Neural Computation* 20, 2629–2636.

Tervo, J., Kolmonen, P., Lyyra-Laitinen, T., Pintér, J.D., and Lahtinen, T. (2003) An optimization-based approach to the multiple static delivery technique in radiation therapy. *Annals of Operations Research* 119, 205-227.

Tilakaratne, Ch.D., Mammadov, M.A. and Morris, S.A. (2008) Predicting trading signals of stock market indices using neural networks. In: Wobcke, W. and Zhang, M., Eds., *AI 2008: Advances in Artificial Intelligence*, pp. 522-531. *Lecture Notes in Computer Science 5360*, Springer-Verlag Berlin Heidelberg, 2008.

Toh, K-A. (1999) Fast deterministic global optimization for FNN training. In: *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics,* Tokyo, Japan, pp. V413 - V418.

Watkin, T. L. H. (1993) Optimal learning with a neural network. *Europhysics Letters* 21 (8), 871-876.

Wikipedia (2010) Artificial neural network. http://en.wikipedia.org/wiki/Artificial_neural_network.

Wolfram Research (2009) *Mathematica* (Version 7.0.1). Distributed by WRI, Champaign, IL.

Ye, H. and Lin, Z. (2003) Global optimization of neural network weights using subenergy tunneling function and ripple search. *Proc. IEEE ISCAS*, Bangkok, Thailand, May 2003, V725-728.