8-2001

Adaptive Simulated Annealing: An Alternative Approach for the Error Minimization of Neural Networks

Yuxing Sun
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Yuxing Sun entitled "Adaptive Simulated Annealing: An Alternative Approach for the Error Minimization of Neural Networks." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Dr. Bruce A. Whitehead, Major Professor

We have read this thesis and recommend its acceptance:

Dr. Kenneth R. Kimble, Dr. Monty L. Smith

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Yuxing Sun entitled "Adaptive Simulated Annealing: An Alternative Approach for the Error Minimization of Neural Networks." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Dr. Bruce A. Whitehead

Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Kenneth R. Kimble

Dr. Monty L. Smith

Accepted for the Council:

Dr. Anne MayHew

Interim Vice Provost and
Dean of The Graduate School

(Orignal signatures are on file in the Graduate Student Services Office)

Adaptive Simulated Annealing: An Alternative Approach for the Error Minimization of Neural Networks

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Yuxing Sun

August 2001

Dedication

This thesis is dedicated to my parents

Xiang Sun

and

Guiying Qu

who have given me inmeasurable love and support.

Acknowledgments

There are many people to whom I am grateful for making my life at Tullahoma easy, and making my time at the University of Tennessee Space Institute rewarding. I would like to thank Dr. Bruce Whitehead who has been my mentor since I arrived at the University, for his patience and guidance to my academic achievements. I would like to thank Dr. Dinesh Mehta for his support over the past many years. I would also like to thank other members in my thesis committee, Dr. Kenneth Kimble and Dr. Monty Smith, for their advice and assistance.

There are a number of other people whose support should be recognized, and they are (in alphabetic order): Dr. Ying-Lin Chen, Dr. Meng Fan, Mehul Kochar, Dr. Ding Li, Bin Liu, Dr. Wenhong Qin, Guoping Xia, and Lin Zhu.

Abstract

This work introduces an alternative algorithm, simulated annealing, to minimize the prediction error from neural networks that traditionally use back-propagation methods. The simulated annealing algorithm stochastically samples the parameter space formed by weights of the neural network until a minimal error is found.

Three problems were investigated in this work: the radiator problem, the spiral problem, and the time series prediction problem. Each of them was examined using the same neural network architecture, i.e., a 2-layer network, and trained by both back-propagation and simulated annealing.

The simulated annealing algorithm consumes longer computation time in searching for the global minimum than the back-propagation method. It may not obtain as small an error as back-propagation in a reasonable amount of time though it theoretically guarantees the location of the global minimum, given enough searching time. It was also found that simulated annealing can more effectively optimize neural networks with a moderate number of weights (less than 32) than those with a large number of parameters, e.g., in the radiator problem, and the spiral problem. Also, simulated annealing is suitable to solve problems with a discrete parameter space.

During the process of tuning the simulated annealing package, configurations with low initial temperatures give faster location of minima, and hence more effective searching, than those with high initial temperatures.

Contents

List of Tables

List of Figures

Nomenclature

$\mathbf{x}$	input vector
x_j	j-th input
x_0	input bias
$\mathbf{w}$	general representation of all weight factors
$\mathbf{w}_i$	weight vector to i-th output node if a hidden layer is present
w_{ik}	k-th component of $\mathbf{w}_i$
$\mathbf{v}_k$	weight vector to k-th hidden node
v_{kj}	j-th component of $\mathbf{v}_k$
y_k^{in}	weighted sum at k-th hidden node
y_k^{out}	output from the k-th hidden node
y_0^{out}	hidden layer bias
z_i^{in}	weighted sum at i-th output node

z_i^{out}	output from the i-th output node
k	annealing time step k
T_k	temperature at time step k
E_k	energy, or cost at time step k
$g_T(\mathbf{w})$	distribution function of the input parameter space at temperature T
$h(\Delta E_k)$	accepting function at time step k

Chapter 1

Introduction

Optimization plays a key role in the development of a variety of engineering systems, for example, determining the optimal routing for a VLSI chip, or the optimal configuration for a telecommunications network. Often, optimization problems require searching through a set of solutions and picking the best one.

Neural networks also require the use of an optimization algorithm to minimize the prediction error. More specifically, the training of a neural network locates a set of weights in such a way that the overall system gives the minimal prediction error. Back-propagation is the most widely used means for optimizing the prediction error on a set of training patterns for the network [3].

However, the back-propagation algorithm can be useful only for systems with the existence of derivatives in the error as a function of the weights. Furthermore, back-propagation uses a gradient search, which finds only a single minimal point of the parameter space. When the

space has multiple minimal points, i.e., a number of local minima with only one global minimum, the search will only find one local minimal point, not necessarily the global minimum. Furthermore, if the parameter space of interest is discrete, gradient search becomes even harder. Thus, stochastic search techniques such as genetic algorithms and simulated annealing have been developed as means of overcoming the local nature of more traditional search methods, and hence, it is an interesting task to study the ability of such stochastic methods to minimize errors in neural networks.

Therefore, the simulated annealing algorithm was investigated as a method for minimizing prediction errors of neural networks. This involved tuning the parameter options of the algorithm to achieve an efficient search, i.e., to obtain a minimal value within a reasonable amount of time. Three training sets were investigated with the implementation: the radiator problem, the spiral problem, and the time series prediction problem.

A publicly available implementation of the simulated annealing algorithm was used to solve the problems. Among all options to be tuned in this implementation, the distribution function, the accepting function, the temperature cooling schedule, and the initial values of all parameters and corresponding temperatures can be defined or adjusted. The approach of using the implementation is to try many combinations of the options, and choose the one that gives the most efficient search and the lowest prediction error.

To compare the results from the back-propagation and the simulated annealing, the same training data set and neural network were used for both cases. As the processes of adjusting program options and minimizing error are distinct between the two algorithms, the approach

taken was to locate the best result for each algorithm, and then to compare the results.

It was found that the back-propagation usually performs a faster search and finds a lower prediction error than simulated annealing in problems with a parameter space of large dimensions, e.g., > 32. It was also found that the choice of low initial parameter temperatures normally gives a more efficient search in the simulated annealing case.

In Chapter 2, the required background will be introduced. Section 2.1 overviews neural networks, Section 2.2 introduces the target problems and shows how back-propagation works, and Section 2.3 introduces the simulated annealing algorithm, as well as its variation with different distribution density functions and cooling schedules. In Section 2.4, an implementation of the adaptive simulated annealing algorithm by Ingber is introduced, and the implementation is used to solve a multi-modal uni-variate function in section 2.5. Chapter 3, Section 3.1 gives a brief description of the implementation of the annealing algorithm for the error analysis of the neural networks. Section 3.2 introduces several implementations that are used to convert state files between neural networks and the adaptive simulated annealing package. Chapter 4 shows the results of the radiator, the spiral and the time series prediction problems. Chapter 5 discusses the results and gives suggestions for the future work.

Chapter 2

Background

2.1 Introduction to Neural Networks

It is well known that the human brain is superior to a computer system at many tasks; for example, human ability to process visual information is much better and faster than the most advanced AI systems. Also, the brain has many desirable features such as being highly parallel, dealing with fuzzy, probabilistic, and inconsistent information, and flexibility in the way that it can adjust to a new environment by "learning".

The computational model of neural networks was originally inspired from neuroscience, and it was aimed at modeling networks of real neurons in the brain. The brain is composed of about 10^{11} neurons. Each neuron has a massive number of connections to other neurons, and such connections are established by a complex chemical process between synaptic junctions between neurons. Usually, the sending end fires a specific transmitter substance when the electrical potential inside its cell surpasses a threshold. It appears that some synaptic junctions of the

sending cell excite the receiving cell, and others inhibit the receiving cell. Learning occurs when modifications are made to the effective coupling between one neuron and another at the junction. Theories about computational neural networks can be found in a number of references[1][2], and only a brief overview is given here.

A typical neural network is composed of neurons with a threshold function, f. The choice of f could be a step function, i.e.,

$$\begin{aligned} f(x) &= -1, \quad x < 0, \\ f(x) &= 1, \quad x > 0, \end{aligned} \tag{2.1}$$

an identity function, i.e.,

$$f(x) = x, \tag{2.2}$$

a Gaussian function, i.e.,

$$f(x) = \exp(-\frac{1}{2}x^2), \tag{2.3}$$

or a sigmoid function, i.e.,

$$f(x) = 1/(1 + \exp(-x)). \tag{2.4}$$

Figure 2.1 shows a typical neuron, i, with a threshold function f, which is evaluated in the right half circle of each neuron. The "dot" in each left half circle represents the weighted summation

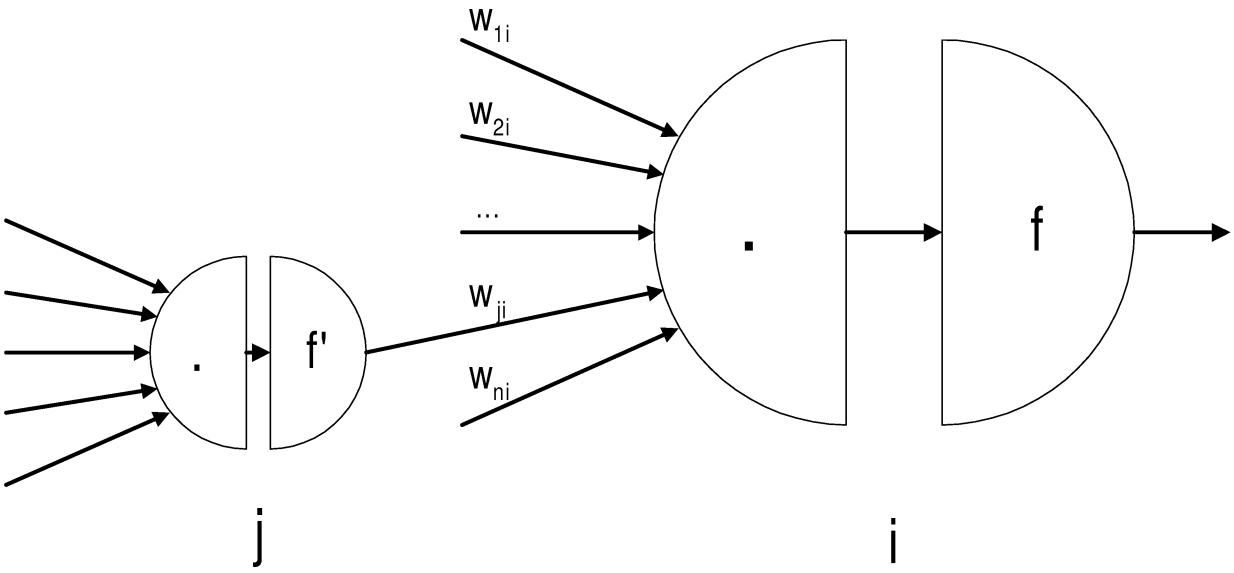

Figure 2.1: Illustration of a Neuron i, with an Input from Neuron j

of all inputs, x_j ($j = 1, 2, \ldots, n$), to the neuron, i.e.,

$$\sum_j w_{ji} x_j + x_0, \tag{2.5}$$

where w_{ji} is the weight of the connection between neuron j and i, and x_0 is a bias term. The resultant summation then passes into the threshold function in order to activate the neurons following it. Each weight w_{ji} can be positive or negative corresponding to an excitatory or inhibitory connections.

Such a neural network model is capable of learning. For example, the network is able to take the inputs provided by users, to feed them through the network, then to compare the calculated outputs with the desired outputs. A prediction error is calculated and some weights of the network are then adjusted for the purpose of minimizing the error. The error is analyzed for each epoch, i.e., for one pass through all input data. The process continues until the minimum error is found.

Typically, all neurons in a network are grouped into layers. The first layer is the input layer, to which users provide inputs for a particular task. The last layer is the output layer, which consists of neurons for outputting results. Some intermediate layers, or hidden layers, can also be used in the network.

The simplest neural network model is a perceptron, and it contains only one node, as illustrated in Figure 2.2. The vectors $\mathbf{x}$ and $\mathbf{w}$ are the input vector and the weight vector, respectively. Thus, the weighted summation, y^{in}, is the dot product of $\mathbf{x}$ and $\mathbf{w}$. In addition, the step function in equation 2.1 is chosen as the threshold function here.

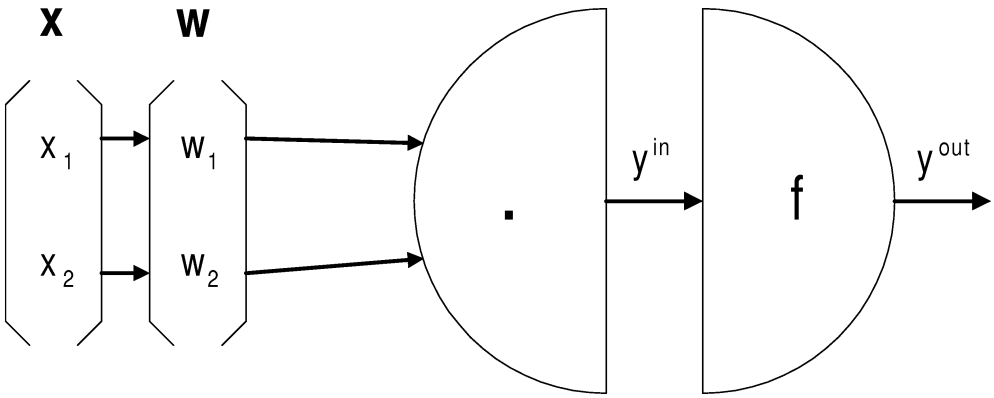

Figure 2.2: Perceptron example: two inputs, one output, and one node

In the learning stage, $\mathbf{w}$ will be changed to $\mathbf{w} + \mathbf{\Delta w}$, where $\mathbf{\Delta w} = -\mathbf{x}$, if the output deviates $+2$ from the desired output, or $\mathbf{\Delta w}$ is equal to $\mathbf{x}$ if the output deviates -2 from the desired output. Hence, the learning rule of the perceptron model can be summarized as

$$\Delta \mathbf{w} = -\frac{1}{2}(y^{out} - y^{desired})\mathbf{x}. \tag{2.6}$$

It is known that neural networks are used for pattern recognition, which is considered to be a two-stage process. The first stage is feature extraction, and the second is classification. Given an input of some form, one can analyze the inputs so that a meaningful categorization of the data content can be obtained. The shortcoming of the perceptron model is that it requires the data set to be linearly separable. For non-linear problems, one has to use a non-linear threshold function, such as a sigmoid function.

The back-propagation algorithm is important to most of the current application work in the field of neural networks. It gives a prescription for changing the weights in a feed-forward type of network. The basis of the algorithm is steepest gradient descent[2].

The introduction of multiple nodes in a single layer, and multiple hidden layers in networks can substantially increase the computational power of neural networks. An example of a network using one hidden layer is shown in Figure 2.3. The network is composed of one hidden node and one output node. x_0 and y_0^{out} in Figure 2.3 are the bias inputs for the hidden and output nodes. The hidden layer weight vector $\mathbf{v}$ and output layer weight vector $\mathbf{w}$ can be varied

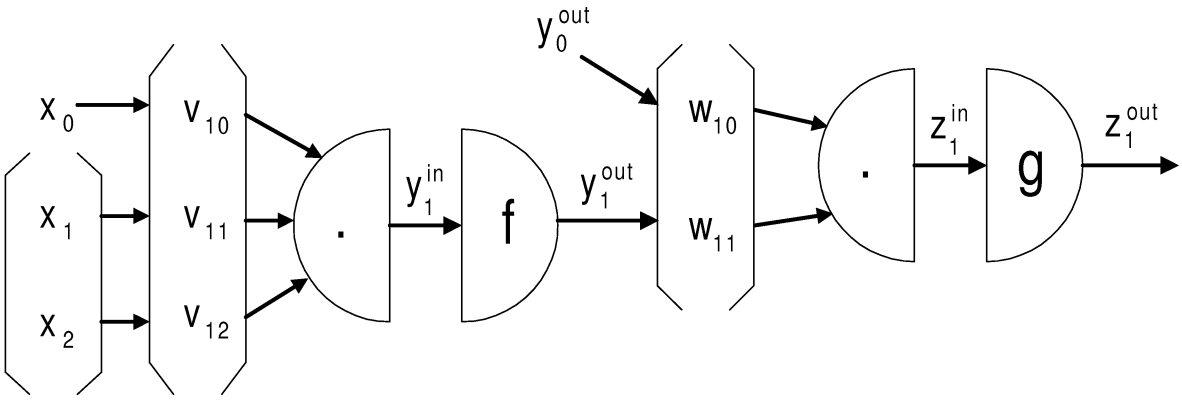

Figure 2.3: Network Architecture for the Radiator Problem

according to the back-propagation learning rule summarized in equations 2.7 and 2.8[3]:

$$\begin{aligned} \Delta \mathbf{w}_i &= -\alpha z_i^{\epsilon} \mathbf{y}^{out}, \\ \Delta \mathbf{v}_j &= -\beta y_j^{\epsilon} \mathbf{x}, \end{aligned} \tag{2.7}$$

where

$$\begin{aligned} z_i^{\epsilon} &= 2(z_i^{out} - z_i^{desired}) g'(z_i^{in}), \\ y_j^{\epsilon} &= \sum_{i=1}^{n} w_{ij} z_i^{\epsilon} f'(y_j^{in}), \end{aligned} \tag{2.8}$$

where the primes in equation 2.8 denote the partial derivative with respect to the variable, and α and β are the learning rates of the output layer and hidden layer, respectively.

Besides the types of networks introduced above, there are also neural network models based on radial basis functions, models with unsupervised learning, such as vector quantizers and topology preserving feature maps, and cascade correlation networks. However, this work will mainly focus on the type of networks discussed above based on inner products and sigmoid functions.

2.2 Examples of Using Neural Networks

In this section, the type of network for studying these examples is selected as the 2-layer network, i.e., a network containing one hidden layer and one output layer, and the learning is governed by the back-propagation rule, given in equation 2.7. It is known that the 2-layer network

is sufficient for arbitrary non-linear systems[4][5][6][7].

The code representing the networks consists of two parts, i.e., user defined code and ebtide library, as illustrated in Figure 2.4. The ebtide library contains the "main" program, which initializes the training process, feeds a fixed set of input data through the network, propagates the error back to each connection and varies each weight accordingly, evaluates the error between calculated and desired outputs after each epoch, and continues iterating until the minimum error is found.

The ebtide library requires the user to provide a learning rule, as well as threshold functions for each layer. These are located in the user defined code, e.g., "*feedback.c*" in the example of Figure 2.4. Note that the user defined code is the only portion of the program that uses a random number generator, which is called to initialize all weights. Normally, a standard *C* function, "*drand48()*", is utilized to fulfill the task. However, if one were to specify the initial random seed for the generator, another *C* function, "*srand(integer_seed)*", has to be called before the first call to "*drand48()*".

Furthermore, a state file, *f1*, is required by the ebtide library to provide information in the initialization stage, for instance, the state file gives the number of inputs, the number of outputs, the number of hidden nodes, and the number of learning epochs. A sample of such a state file is shown in Figure 2.5.

The output file from the code has a similar format as the state file *f1* except that all the weights in the file will be assigned their final values which are obtained after training has been completed in the specific number of epochs.

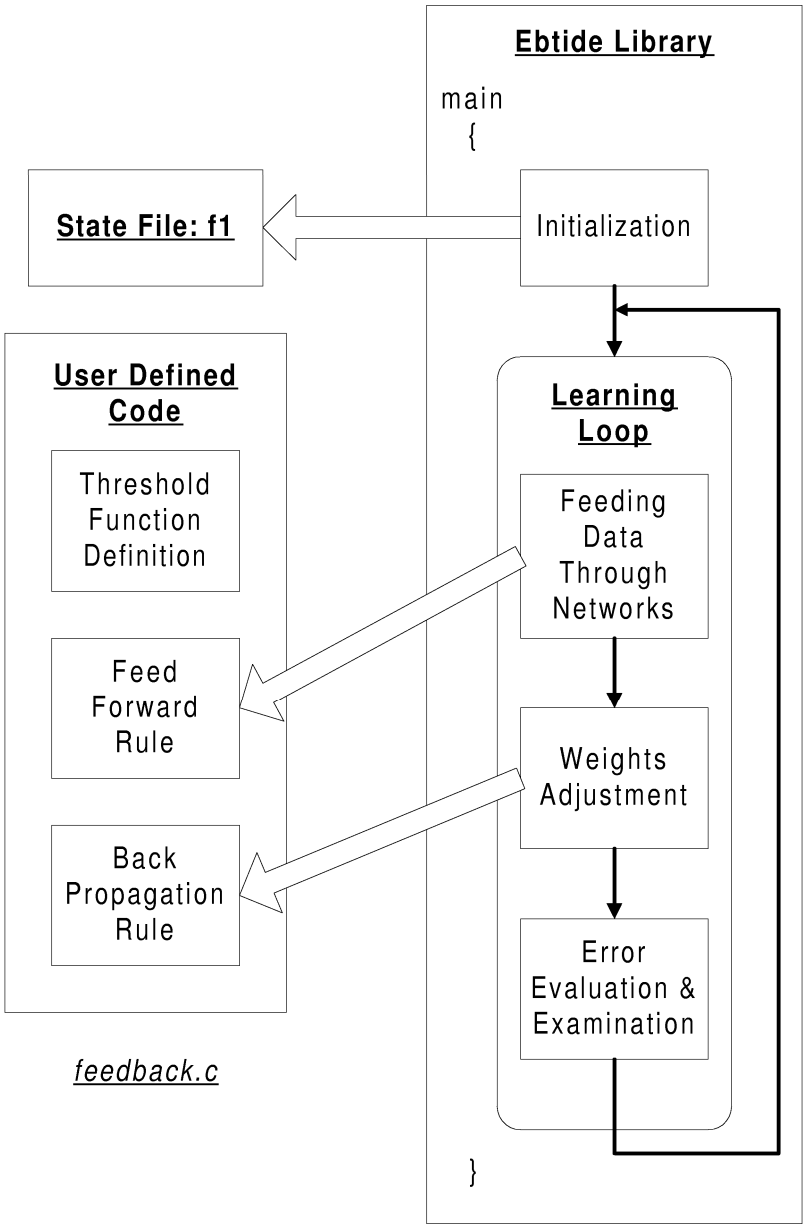

Figure 2.4: Main Components of Neural Network Analysis

```
model_state [
n_inputs 2
n_outputs 1
n_hiddenA 1
epoch 0
epoch_to_stop 10000
plot_interval 1000
little_report_interval 1000
big_report_interval 1000
print_databuf_interval 1
learning_rate 0.1
output_transfer_function_name sigmoid
output_transfer_deriv_name sigmoid_deriv
hiddenA_transfer_function_name sigmoid
hiddenA_transfer_deriv_name sigmoid_deriv
... ...
hidden_learning_rate 0.25
... ...
]
```

Figure 2.5: State File Sample, Required by the Ebtide Library

2.2.1 The Radiator Problem

The first example of using neural networks is a simple example taken from the class lectures of "Cybernetics", offered by Dr. Bruce Whitehead during fall semester of 1999 and spring semester of 2000. It is a two dimensional linearly separable problem, and the given training data are drawn in Figure 2.6. Each point in the figure hypothetically represents the working condition inside a vehicle engine, with the X axis as temperature and the Y axis as pressure. The solid and hollow circles in Figure 2.6 stand for normal and abnormal conditions of the engine, respectively. The results obtained for this example would be relevant to real applications whose inputs are linearly separable or nearly so. The remaining two problems, the spiral problem and the time series prediction problem, are intended to represent more difficult problems, in which the output depends upon non-linear interactions among the input variables.

The network shown in Figure 2.3 is chosen to solve this problem. The hidden layer and the output layer each contain only one node, and hence there are only 5 weights in the network. The threshold functions for the hidden and output nodes are chosen as the sigmoid function and the identity function, respectively. The training results will be shown in Chapter 4, Section 4.1.

2.2.2 The Spiral Problem

In the spiral problem [8], each data set contains 2 inputs and 1 output. If two inputs are taken as the variables of a two dimensional plane, all points will form two intertwined spirals, as shown in Figure 2.7. The task of the neural network is to output -0.5 if the test point falls on the lower spiral, or 0.5 if it falls on the other. Because the spiral problem requires the network to

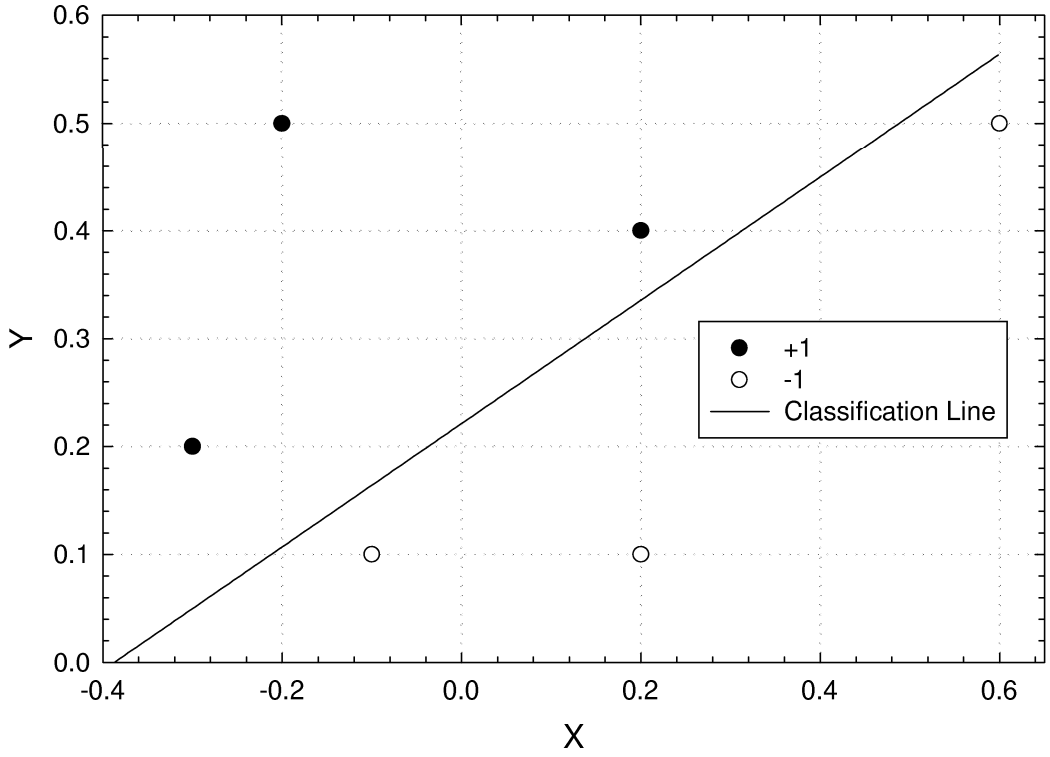

Figure 2.6: Input Data Set for the Radiator Problem

learn a highly non-linear separation of the input space, and its 2-dimensional input space makes it easy to plot the network's transfer function in order to study the network's inner workings and development during learning, it is considered as an interesting benchmark task for neural networks. It is intended to be representative of the class of classification problems in which discrete categories are discriminated by non-linear combinations of input variables.

The network shown in Figure 2.8 is constructed to solve the problem. For the purpose of comparing results from both the back-propagation and the adaptive simulated annealing, to be introduced in section 2.3, the same network structure is used for both the cases. In the network shown in Figure 2.8, The hidden layer is chosen to have 8 nodes, and the threshold function for these nodes are the sigmoid function. The reason for using 8 hidden nodes is that fewer nodes give poor prediction error, around 0.8–0.9, and choosing more than 8 hidden nodes will substantially increase the search time in the adaptive simulated annealing case. The threshold function for the output layer is selected to be the identity function, given by equation 2.2. The choice of the threshold function in the output layer is arbitrary, and the sigmoid function can also be used.

The results from both the back-propagation and the adaptive simulated annealing will be given in Chapter 4, Section 4.2.

2.2.3 The Time Series Prediction Problem

The time series prediction problem[9][10] is recognized as a benchmark for comparing the learning and generalization ability of different neural architectures, and it originated from a

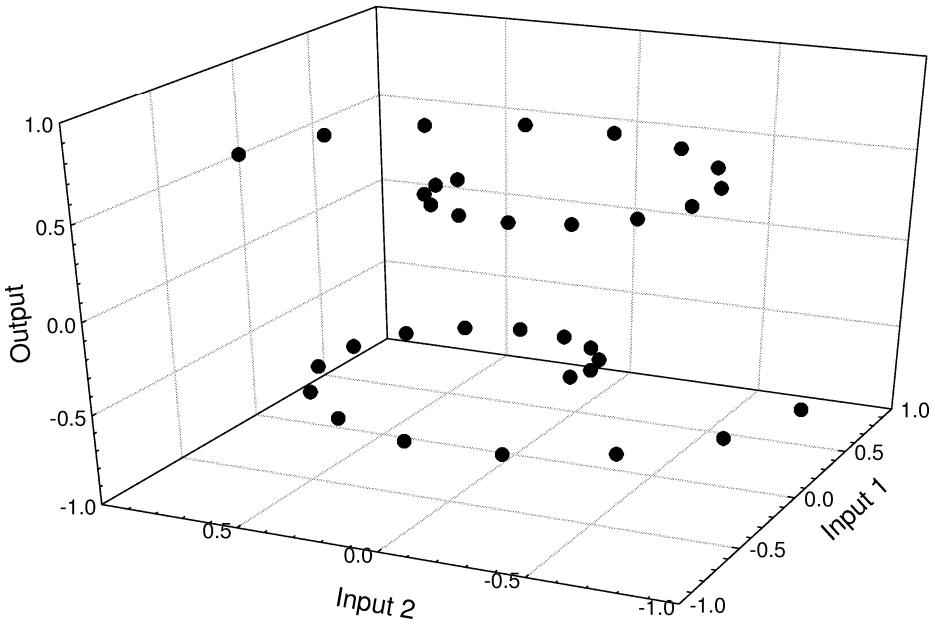

Figure 2.7: Training Set of the Spiral Problem

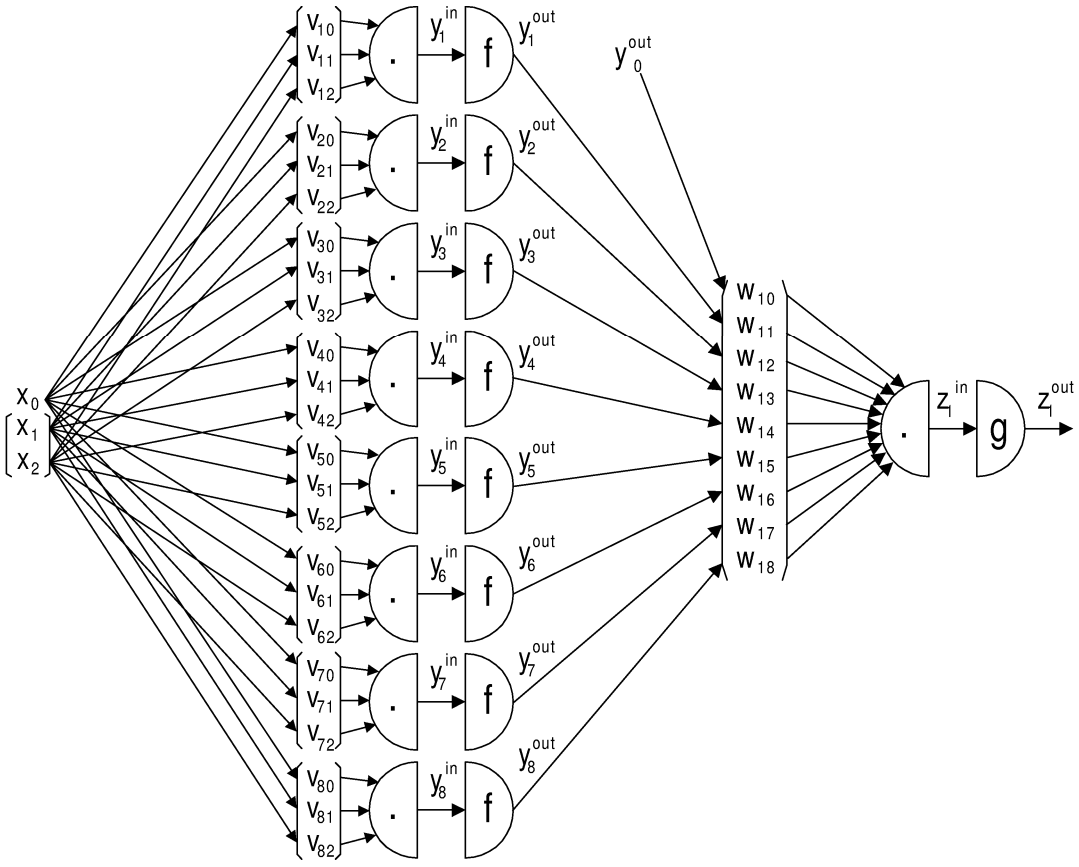

Figure 2.8: Architecture of the Networks for Solving the Spiral Problem

number of chronic and acute diseases researches. Mackey *et al.*[9] studied the first order non-linear differential-delay equations in order to describe physiological control systems, and to show that simple mathematical models of physiological systems predict the existence of regimes of periodic and aperiodic dynamics. This problem is intended to be representative of applications, in which the output variable is continuous and is highly non-linear function of the input variables. The problem is based on the Mackey–Glass differential equation, i.e.,

$$\frac{dx(t)}{dt} = -bx(t) + a \cdot \frac{x(t-T)}{1 + x(t-T)^{10}}, \tag{2.9}$$

where $a = 0.2$, $b = 0.1$ and $T = 17$. The task for the neural network is to predict the value of the time series at time $t + I$ from the earlier points $x(t)$, $x(t - D)$, $x(t - 2D)$, and $x(t - 3D)$, where $I = 75$ and $D = 6$. Therefore, each data set contains 4 inputs and 1 output.

For this problem, a neural network is constructed as shown in Figure 2.8 except that it has a hidden layer with 16 nodes. The reason for using 16 hidden nodes is that fewer nodes give poor prediction error, around 0.8–1.0, and choosing 16 or more hidden nodes substantially increases the search time in the adaptive simulated annealing case. The threshold functions for both hidden and output layers are chosen as the sigmoid function. The results from the time series prediction problem will be given in Chapter 4, Section 4.3.

2.3 Introduction to Adaptive Simulated Annealing

2.3.1 Overview

In the early 1980's, Kirkpatrick *et al.*[13] introduced the concepts of annealing in combinatorial optimization. These concepts are based on an analogy between the physical annealing process of solids and the problem of solving large combinatorial optimization problems.

In condensed matter physics, annealing is known as a thermal process for obtaining low energy states of a solid in a heat bath. The process begins with increasing the temperature of the heat bath to a maximum value at which the solid melts, then decreasing the temperature carefully until the particles arrange themselves in the ground state of the solid. In such case, the particles form a highly structured lattice and the energy of the system is minimal.

A computer simulation of the physical process of annealing was done by Metroplis *et al.*[14] in 1953, and it is known as the Boltzmann annealing. The algorithm of the simulation generates a sequence of states of the solid by Monte Carlo techniques. Given the current state i and corresponding energy of the solid, E_i, a subsequent state j, with energy E_j is generated. State j is generated by a small distortion in state i. If the energy difference, $E_j - E_i$, is less than or equal to 0, the state j is accepted as the current state, otherwise, the state j is accepted with a certain probability, given by

$$\exp(-\frac{E_j - E_i}{k_B T}), \tag{2.10}$$

where T is the temperature of the simulated heat bath and k_B is the Boltzmann constant. If the lowering of the temperature is done sufficiently slowly, the solid can reach thermal equilibrium

at each temperature. This equilibrium is governed by the Boltzmann distribution[11], given by

$$\mathbf{P}(i,T) = \frac{1}{Z(T)} \exp(-\frac{E_i}{k_B T}), \tag{2.11}$$

where in the equilibrium condition, $\mathbf{P}(i,T)$ denotes the probability of generating a state i at temperature T, and $Z(T)$ is the partition function, which is defined as

$$Z(T) = \sum_j \exp(-\frac{E_i}{k_B T}). \tag{2.12}$$

The simulated annealing algorithm can be used to search for the minimal error of a neural network if one assumes an analogy between a physical many-particle system and an optimization problem. Given a target function that is to be optimized, the annealing state is determined by the current temperature and its location in the parameter space, i.e., the space formed by the parameters of the function to be optimized. The function to be optimized under such condition corresponds to the energy of the solid. The acceptance criteria of each generated state is based on equation 2.10. The transition between states is realized as a transition of locations in the parameter space. Thus, the probability of generating the next state in the parameter space is governed by equation 2.11 in Boltzmann annealing. In the problem of error minimization of neural networks, the target function, or cost function, is the one that evaluates normalized error of the feed forward type of network. Note that the cost function generally depends on all weights and input data set. However, a fixed set of input data is used in this work, and hence the cost function is only determined by all the weights.

2.3.2 Theory of Simulated Annealing Methods

Systematically, the simulated annealing algorithm is composed of three functional relationships, i.e., the generating probability density function of a space of D parameters, $g_T(\mathbf{x})$, the acceptance probability density function of accepting a new cost function value given the difference from the previous value, $h(\Delta E)$, and the cooling schedule of the temperature $T(k)$ at time step k. A point in the D dimensional parameter space is denoted by a vector $\mathbf{w} = (w^0, w^1, \ldots, w^D)$.

In Boltzmann annealing[12], the acceptance probability based on the chance of obtaining a new state with energy E_{k+1} relative to a previous state with energy E_k as given by equation 2.10, with subscript i replaced by k and j replaced by $k + 1$.

The algorithm also can be described by considering a set of states labeled by $\mathbf{w}$, each with energy $E(\mathbf{w})$. Therefore, under the thermal equilibrium, the Boltzmann distribution governs the generating probability density[12], and equation 2.11 can be expressed as

$$g(\Delta\mathbf{w}) = (2\pi T)^{-D/2} \exp[-\Delta w^2/(2T)], \tag{2.13}$$

in D-dimensional space, and $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_0$ is the deviation of $\mathbf{w}$ from the previously chosen point, $\mathbf{w}_0$.

Given $g(\Delta\mathbf{w})$, it has been proven[15] that it suffices to obtain a global minimum of the cost function, $E(\mathbf{w})$, if the cooling schedule of T is selected to be not faster than

$$T(k) = T_0/ln(k). \tag{2.14}$$

2.3.3 Variation of the Boltzmann Temperature Schedule

The standard Boltzmann annealing algorithm is able to handle quite complex cost functions and constraints. However, due to the drawback of its slow search, a faster temperature cooling schedule is preferred, and the schedule in equation 2.14 may be replaced by

$$T_{k+1} = cT_k, \tag{2.15}$$

where $0 < c < 1$. This is also called an exponential cooling schedule. The algorithm with the generating probability density function, $g_T(\mathbf{w})$, and the acceptance probability density function, $h(\Delta E)$, from the standard Boltzmann annealing, yet using a faster exponential cooling schedule is known as simulated quenching[12]. Such a process is the converse of annealing, in which the temperature of the heat bath is instantaneously lowered. This will result in a meta-stable state, and it does not guarantee location of the global optimal point.

Others[16] found that, by replacing Boltzmann distribution with Cauchy distribution, one can speed up the search, and the algorithm is hence called fast annealing. The generating probability density function is defined as

$$g(\Delta\mathbf{w}) = \frac{T}{(\Delta w^2 + T^2)^{(D+1)/2}} \tag{2.16}$$

for the reason that the Cauchy distribution has a fatter tail than that of the Boltzmann distribution, permitting easier access to test local minima in the search for the desired global minimum.

Under such circumstances, the cooling schedule is given by

$$T(k) = T_0/k, \tag{2.17}$$

and the algorithm is heuristicly demonstrated to be able to locate the global minimum[12].

The last variation of the standard Boltzmann annealing is the adaptive simulated annealing(ASA) algorithm, and it was the main algorithm used in this work. It deals with a D-dimensional space with parameters w_k^i $(i = 1, \ldots, D)$, each of which ranges from lower bound A_i to higher bound B_i, in which k denotes the annealing time step. The next location of the searching route is generated by

$$w_{k+1}^i = w_k^i + y^i(B_i - A_i), \tag{2.18}$$

where y^i is a random variable within $[-1, 1]$.

A global minimum statistically can be obtained if the generating probability density function for each y^i is given by

$$g_T(\mathbf{y}) = \prod_{i=1}^{D} \frac{1}{2(y^i + T_i)ln(1 + 1/T_i)}, \tag{2.19}$$

where i on temperature T_i specifies the parameter index, and the cooling schedule of each parameter is defined by

$$T_i(k) = T_{0i} \exp(-c_i k^{1/D}), \tag{2.20}$$

where T_{0i} is the current initial temperature for the i-th parameter.

Note that c_i in equation 2.20 is further defined by the (m_i, n_i) pair in the ASA package, introduced in Section 2.4, and the following relations hold at time step k, i.e.,

$$T_i(k) = T_{0i} \exp(-m_i), \tag{2.21}$$

$$k = \exp(n_i), \tag{2.22}$$

and

$$c_i = m_i \exp(-n_i/D). \tag{2.23}$$

The (m_i, n_i) pair is a parameter option used to tune the ASA algorithm for specific problems.

Another adaptive feature of ASA is its ability to perform quenching by just re-replacing the cooling schedule in equation 2.20 with

$$T_i(k) = T_{0i} \exp(-c_i k^{Q_i/D}) \tag{2.24}$$

and replacing equation 2.23 with

$$c_i = m_i \exp(-n_i Q_i/D), \tag{2.25}$$

where Q_i is the quenching factor.

In the ASA algorithm, the influence of large dimensions is from the exponential power term $1/D$ of the time step k. The execution becomes prohibitively slow as the annealing attempts to sample the parameter space. The quenching factor can be turned on in the case of many

dimensions, however it does not guarantee that the global minimum can be located.

2.4 The ASA Implementation

The ASA algorithm has been implemented by Lester Ingber[12], and the source code is publicly available. The code is written in the *C* language, and it is kept updated and new features have been added to the package since 1993. The most current version, "v.20.2", of the implementation, as well as some research papers concerning ASA algorithms, can be found at the web-site "http://www.ingber.com". This work mainly uses the version "v.17.19" of the ASA package.

2.4.1 Components of the ASA Package

The current version of the ASA package contains two basic modules, i.e., the user module and ASA module. The user module includes 3 files, i.e., *user.h*, *user.c*, and *user_cst.h*. The target cost function can be implemented in the package by changing part of the code in *user_cst.h*, as shown in Figure 2.9. The ASA module is composed of 2 files, i.e., *asa.h* and *asa.c*. There is also an option input file, *asa_opt*, which is read by the user module at runtime, and this file contains parameters that can be used to tune the ASA process.

Random numbers in the ASA package are generated as described in reference[17]. The algorithm uses an initial seed, $rand\_seed = 696969$, which could be changed into other arbitrary numbers. In Chapter 4, such arbitrary numbers are chosen as 111111, 222222, ..., 999999.

```
double cost_function (double *x,
          double *parameter_lower_bound,
          double *parameter_upper_bound,
          double *cost_tangents,
          double *cost_curvature,
          ALLOC_INT * parameter_dimension,
          int *parameter_int_real,
          int *cost_flag,
          int *exit_code,
          USER_DEFINES * USER_OPTIONS)
{
*  double energy = 0.0;
*  int i;
*  for(i = 0; i < *parameter_dimension; i++)
*    {
*      energy += x[i]+10.0*sin(5.0*x[i])+7.0*cos(4.0*x[i]);
*    }
*  *cost_flag = TRUE;
*  return energy;

#if ASA_TEST
  double q_n, d_i, s_i, t_i, z_i, c_r;
... ...
}
```

Figure 2.9: Insertion of Cost Function in File *user_cst.h*, as shown in lines starting with $*$

2.4.2 Output Files

The ASA code normally has its outputs written into two files, i.e., *asa_out* and *user_out*. The latter contains only the resultant values of the cost function and all parameters. The *asa_out* file outputs information about the searching process and is controlled by the pre-compiled print options. It usually starts with a list of all options of the execution, then prints all the accepted states, as well as each corresponding generated state and the corresponding value of the cost function. Samples of file *asa_out* and *user_out* are shown in Figure 2.10 and 2.11.

2.4.3 ASA Options

The *asa_opt* file contains most of the important program options that can be changed without re-compiling the code. The file gives the initial temperature, the parameters controlling the cooling schedule, the precision of calculation, and some printing options, etc., then it defines the total number of parameters, and their corresponding ranges. Alternatively, the options can also be changed inside of *Makefile* and *user.c*, but this requires the code to be re-compiled. A sample of *asa_opt* file is shown in Figure 2.12

2.4.4 Main Options

In the process of tuning the ASA algorithm for this work, 5 program options were found to be crucial to the final outcomes. They will be described in detail in this section.

```
OPTIONS_FILE = 1
OPTIONS_FILE_DATA = 1
RECUR_OPTIONS_FILE = 0
RECUR_OPTIONS_FILE_DATA = 0
COST_FILE = 1
... ...
OPTIONS->Limit_Acceptances = 100000
OPTIONS->Limit_Generated = 999999
OPTIONS->Limit_Invalid_Generated_States = 1000
OPTIONS->Accepted_To_Generated_Ratio =        1e-06
... ...
*number_parameters = 5

index_v parameter_minimum parameter_maximum parameter_value
parameter_type
temperature_scale =     8.716998
temperature_scale_parameters[0] =     8.716998
*temperature_scale_cost =     8.716998

best...->cost=9.027746      *number_accepted=1  *number_generated=1
best...->cost=5.603044      *number_accepted=2  *number_generated=2
best...->cost=4.227825      *number_accepted=3  *number_generated=3
best...->cost=4.201684      *number_accepted=4  *number_generated=5
best...->cost=2.276026      *number_accepted=5  *number_generated=7
... ...
best...->cost=0.007272216   *number_accepted=69  *number_generated=1396
best...->cost=0.005837542   *number_accepted=70  *number_generated=1406
best...->cost=0.005062919   *number_accepted=71  *number_generated=1580
best...->cost=0.004603576   *number_accepted=72  *number_generated=1585
best...->cost=0.00460023    *number_accepted=73  *number_generated=1949
best...->cost=0.004591624   *number_accepted=74  *number_generated=2129
best...->cost=0.004512939   *number_accepted=75  *number_generated=2225

*index_cost_acceptances = 3, *current_cost_temperature = 8.678426e-08
*accepted_to_generated_ratio =  0.001004016, *number_invalid... = 0
*number_generated = 3220, *number_accepted = 75
best...->cost =  0.004512939, last...->cost =  0.004512939
index_v  best...->parameter current_parameter_temp   tangent
0        -18.53141           1.333859e-05       0.000356695
1         -47.8606           1.333859e-05      0.0003298993
2         84.06385           1.333859e-05      0.0002192651
3     -0.002072142           1.333859e-05      0.0004494491
4         2.011917           5.186192e-16          0.183155
... ...
 COST_REPEATING exit_status = 3
 Locate_Cost = 5, calculating curvatures while exiting asa ()
final_cost = best_generated_state->cost = 0.004512939
*number_accepted at best_generated_state->cost = 75
*number_generated at best_generated_state->cost = 2225
```

Figure 2.10: Sample of ASA_OUT

```
exit code = 0
final cost value =  0.004512939
parameter  value
0          -18.53141
1           -47.8606
2           84.06385
3       -0.002072142
4           2.011917
```

Figure 2.11: Sample of USER_OUT

```
Limit_Acceptances[10][ASA_TEST:1000]                    100000
Limit_Generated[99999]                                  999999
Limit_Invalid_Generated_States[1000]                    1000
Accepted_To_Generated_Ratio[1.0E-6][ASA_TEST:1.0E-4] 1.0E-6
Cost_Precision[1.0E-18]                                 1.0E-50
Maximum_Cost_Repeat[5]                                  100
Number_Cost_Samples[5]                                  50
Temperature_Ratio_Scale[1.0E-5]                         1.0e-6
Cost_Parameter_Scale_Ratio[1.0]                         1.0
Temperature_Anneal_Scale[100.0]                         10.0
Include_Integer_Parameters[FALSE=0]                     0
User_Initial_Parameters[FALSE=0]                        0
Sequential_Parameters[-1]                               -1
Initial_Parameter_Temperature[1.0]                      10.0
... ...
number_parameters=*parameter_dimension                  5
Param#:Minimum:Maximum:InitialValue:Integer[1or2]orReal[-1or-2]
0   -100    10  0.999          -1
1   -100    20  -1.007         -1
2   -100    100 1.001          -1
3   -10     10  -1.803         -1
4   -10     10  0.999          -1
... ...
```

Figure 2.12: Sample of ASA_OPT

Initial Cooling Temperature

The "Initial_Parameter_Temperature" option sets initial temperatures for all the parameters, and its default value is 1.0. By tuning this option to a higher value, the initial searching step of the algorithm becomes fairly big, and might result in traversing a large portion of parameter space.

Factors Controlling Cooling Schedule

Recall that the cooling schedule of the ASA algorithm is controlled by a (m_i, n_i) pair, as described in section 2.3.3. Two options are directly related to the pair, i.e., "Temperature_Ratio_Scale" and "Temperature_Anneal_Scale". By default, each m_i and n_i is given by

$$m_i = -\log(\text{Temperature\_Ratio\_Scale}) \tag{2.26}$$

and

$$n_i = \log(\text{Temperature\_Anneal\_Scale}). \tag{2.27}$$

It was found that proper use of the combination of two options is crucial for obtaining the optimal results in a reasonable amount of time.

Cost vs. Parameter

"Cost_Parameter_Scale_Ratio" is another major factor that influence the searching efficiency of the algorithm, and its default value is 1.0. It is the ratio of temperature annealing scales between

the cost and parameter, i.e.,

$$c_{\text{cost}}/c = \text{Cost\_Parameter\_Scale\_Ratio}. \tag{2.28}$$

It was found to be sufficient to keep this option under 5.

Number of Cost Samples

The "Number_Cost_Samples" option defines the number of the samples that are used to evaluate the average of the cost function. It was found that this option is sensitive to the searching step of the algorithm. A small number of samples gives a more detailed search, however, in such cases, the algorithm more easily trapped in a local minimum.

2.4.5 Other ASA Options

Limit_Generated

This option determines the upper limit of the total number of the generated states, and it is effective for controlling the timing of the exit of the search.

Limit_Acceptances

This option controls the upper limit of the total number of the acceptance states. If the number of accepted states surpasses the defined limit, the search will exit. This option is usually set to be comparable to the "Limit_Generated" to avoid an early exit.

Accepted_To_Generated_Ratio

This option sets up the ratio between the current number of the accepted state and the generated state, and its value is always set to be small so as to avoid early exit.

Cost_Precision

The smallest floating point number is defined by this option. If the value is set to be large and the initial temperature of the schedule is comparably low, the algorithm usually ends up with an early exit.

Maximum_Cost_Repeat

It happens quite often that the value of the cost function repeats a number of time when the search is trapped in a local minimum. It would be better to choose a fairly large number that is comparable to the limit of the number of the accepted states to avoid un-nessesary exit from the searching algorithm.

User_Initial_Parameters

This option needs to be set to TRUE when the initial value of parameters, which are provided by user, need to be evaluated.

Printing Options

There are several printing options that can be set to print out more information about the search. For example, setting “ASA_PRINT_INTERMED” and “ASA_PRINT_MORE” to TRUE will

Table 2.1: Program Options for Fast Quenching Algorithm

Option	Value	Option	value
USER_REANNEAL_PARAMETERS	TRUE	USER_COST_SCHEDULE	TRUE
ASA_PRINT_INTERMED	FALSE	SMALL_FLOAT	1×10^{-50}
QUENCH_PARAMETERS	TRUE	QUENCH_COST	TRUE
QUENCH_PARAMETERS_SCALE	FALSE	QUENCH_COST_SCALE	FALSE
Curvature_0	TRUE	Temperature_Ratio_Scale	$\downarrow$
Cost_Parameter_Scale_Ratio	$\downarrow$	Maximum_Cost_Repeat	$\uparrow$
Acceptance_Frequency_Modulus	$\downarrow$	Generated_Frequency_Modulus	$\downarrow$

give a detail description after each epoch of searching. However, it is found that such information is not always helpful to obtain hints for a better and faster search, but activating these options will substantially slow down the execution of the code.

In addition, there is a convenient way to number the output *asa_out* and *user_out* files by setting ASA_OUT=\"asa_out_\" and USER_OUT=\"user_out_\" options, and this is done by acquiring the execution process ID from the operating system.

All the printing options can be configured from the *Makefile*.

Fast Quenching Options

The author of the ASA package gave suggestions for searching very high dimension parameter spaces[18], and basic idea is to use the fast quenching, introduced in section 2.3.3. Under such circumstances, some of the options are chosen as shown in Table 2.1. where $\downarrow$ or $\uparrow$ represents that the specific option is decreased or increased comparing to its default setup value.

Nevertheless, it was found that this setting of options did not give a more efficient search than the default options for the problems at hand, and it does not guarantee to locate the global

Table 2.2: Program Options in *asa_opt* for Solving Multi-modal Uni-variate Function

Option	Value	Option	value
Limit_Acceptances	10000	Limit_Generated	99999
Limit_Invalid_Generated_States	10000	Accepted_To_Generated_Ratio	10^{-6}
Cost_Precision	10^{-40}	Maximum_Cost_Repeat	1000
Number_Cost_Samples	5	Temperature_Ratio_Scale	0.9
Cost_Parameter_Scale_Ratio	1.0	Temperature_Anneal_Scale	1.0
Include_Integer_Parameters	FALSE	User_Initial_Parameters	TRUE
Sequential_Parameters	-1	Initial_Parameter_Temperature	100.0
Acceptance_Frequency_Modulus	10	Generated_Frequency_Modulus	100
Reanneal_Cost	1	Reanneal_Parameters	TRUE
Delta_X	0.1	User_Tangents	FALSE
Curvature_0	FALSE	*parameter_dimension	1
Param#:Minimum	0.0	Param#:Maximum	10.0
ASA_TEST	FALSE		

minimum.

2.5 An Example of Using ASA

As traditional gradient optimization algorithms normally tend to locate only one minimum of a multi-modal function, not nessesarily the globally optimal point, the following example is presented to illustrate the ASA search procedure.

An arbitrary multi-modal function, $y = x + 10sin5x + 7cos4x$, was chosen to test the ASA package, as shown in Figure 2.13. The search has been conducted from two different initial points, i.e., $x = 0.0$ and $x = 10.0$. The configuration of the program options in *asa_opt* and *Makefile* is listed in Table 2.2.

For both the cases, the minimum value of -15.1644 is found at $x = 0.8917489$, and the process shows that the ASA package is able to locate the global minimal point after generating

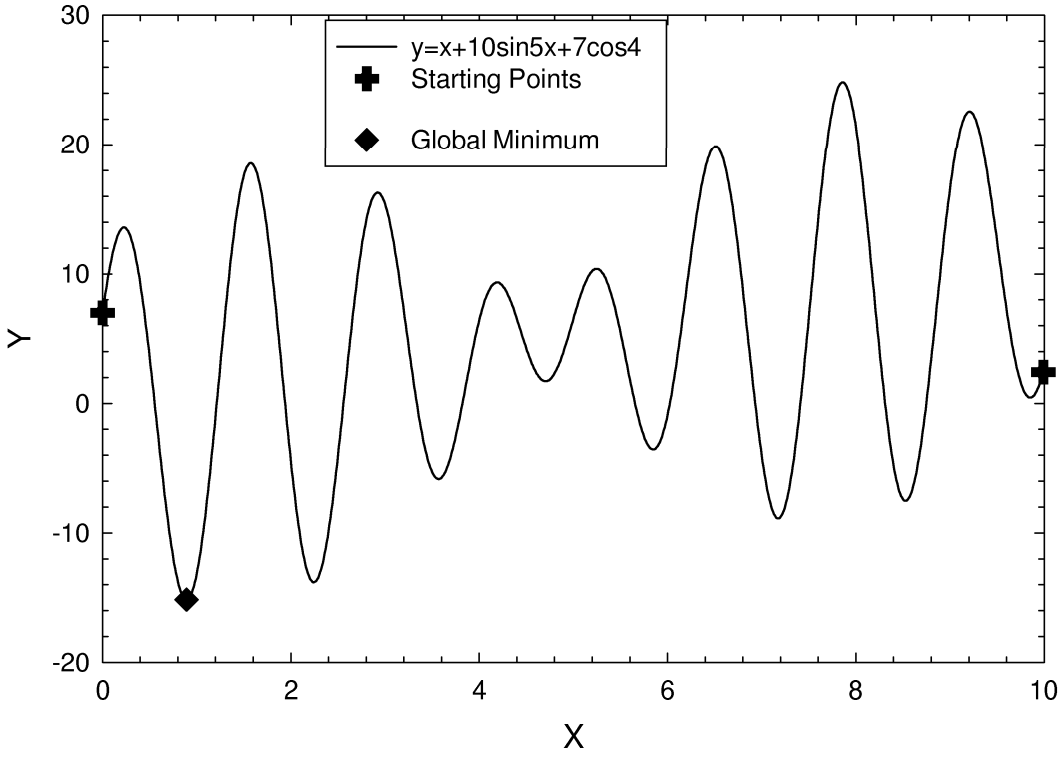

Figure 2.13: A Multi-modal Uni-variate Function

about 60 states.

Chapter 3 will introduce some of the work of implementing the ASA package into the error minimization of neural networks.

Chapter 3

Implementation

3.1 Code Implementation

As introduced earlier in section 2.3.1, the error minimization that is traditionally done by back-propagation can alternatively be conducted by the ASA algorithm, in which case the evaluation of the normalized error plays the role of the cost function, where the parameters represent all the weights in the network.

The changes made to the code of the neural network are shown in Figure 3.1. The revised version of ebtide library performs only the task of feeding one epoch of input data through the network, then evaluating the error and returning the result back to the ASA package. The modification to the *user_cst.h* file is similar to the one shown in Figure 2.9 except that a function call to the ebtide library replaces the lines of code starting with $*$.

Note that the *main* program is located at *user.c* file after the changes above. A new *Makefile* is also constructed to keep both the features from ebtide library and the ASA package, For

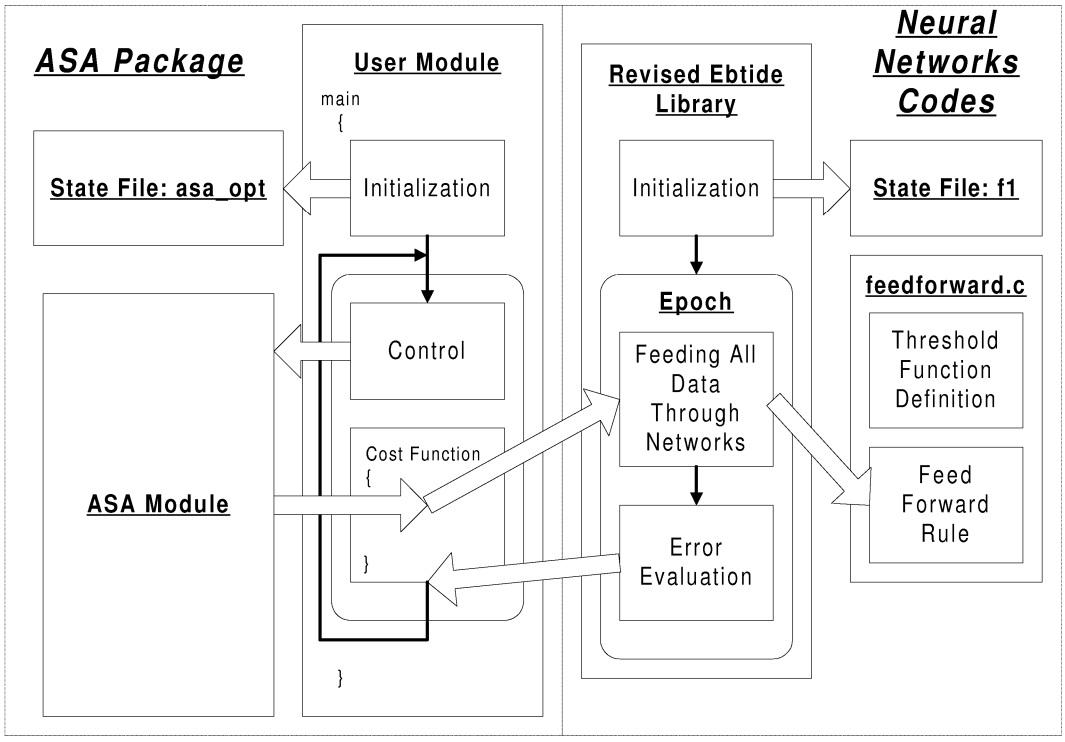

Figure 3.1: Architecture of Package Combining ASA and Neural Networks Analysis

example, user is able to set ASA options from the *Makefile*. However, one has to modify a line of code in *ebtide_objective.c* to name the source files of training or testing data.

The procedure for use of the implemented package is as follows:

- construction of a standard state file *f1*, similar to the one used in section 2.2.1, as the initial state of the ebtide library,
- choosing proper options in *asa_opt*, and also setting ranges for each parameter (Note that default setting of the options would be sufficient in most of the cases, and this work will keep all options as their defaults if no specific changes are noted),
- Compiling the code, and re-compiling is un-nessesary when only *asa_opt* is modified,
- Further tuning all options and re-running the program if nessesary,
- Studying the searching process by illustrating cost function vs. generated state curve,
- Studying the consistency of the resultant weights by plugging all weights back into the neural network with the same structure, and checking if the network gives the same minimal error.

3.2 Implementation of Conversion Algorithms between State Files

It appears rather tedious to transfer the resultant state file of the ASA package into a state file required by the ebtide library for a problem with large number of parameters. Also, the same problem occurs when user wishes to take good training results from back-propagation

as the initial parameters for the ASA searching. The following sections will introduce several implementations for conversions among various state files.

3.2.1 Conversion From Ebtide Library State file to ASA Option File

It is usually helpful for the ASA search to start with a set of parameters that have already given relatively low cost. For example, a set of parameters resulting from back-propagation training could be used as the initial point for ASA searching. Knowledge would be gained if one were to restrict the parameter ranges to a tiny hyper-cube around the initial point, and to tune the program options in order to retain the same optimal result when the ranges are extended.

Therefore, a program was written for the purpose of conversion from the ebtide state file into an ASA option file, i.e., *asa_opt*. The program is named as *n2a.java*, and is coded in the *Java* language. The routine starts to read in all resultant parameters from a ebtide state file, e.g., x_i, $i = 0, 1, 2, \ldots, N$, and determines the precision of each dimension by observing its right most significant digit, i.e., Δx_i for x_i. The range of the i-th parameter is then obtained as $x_i \in [x_i - a\Delta x_i, x_i + a\Delta x_i]$, where a can be chosen as 2, 4, 8, ..., 1024, ..., in order to form various sizes of hyper-cubes.

The *n2a.java* requires two variables in the command line. One is an output state file from the neural network training, and this file is required to contain values for all parameters. The second variable is a standard *asa_opt* file, which does not need to have same number of parameters as in the ebtide state file.

Another version of this program determines a universal range for all parameters. The range

is chosen to cover the minimal and maximal parameters in the ebtide library. This implementation is called *spiral.java*, and it requires the same number of variables from the input command line as *n2a.java*.

3.2.2 Conversion From ASA User Output File to Ebtide Library State file

Conversely, the resultant parameters from the ASA output can also be converted into a state file of ebtide library.

The program written for this purpose is *state.java*. This routine requires three variables from the input command line. The first is the name of a *user_out* file from ASA output, the second is a standard ebtide state file, which needs to have the correct number of inputs, outputs and hidden nodes of the desired network, and the last variable is the name for the output ebtide state file name. Also, a correct definition of the total number of parameters is required inside of the *state.java* code.

3.2.3 Conversion From ASA User Output File to ASA Option File

It is also convenient for users to be able to convert ASA output directly into an ASA option file, i.e., from *user_out* to *asa_opt*. This will help to perform one ASA search starting from the end point of the preceding search.

The program is called *track.java*, and it requires 2 variables from the input command line. The first one is the name of the *user_out* file, and the second is a standard *asa_opt* file, which does not need to have same number of parameters as in the ASA output file. One limitation of the this routine is that one has to provide the number of inputs, outputs and hidden nodes inside

of the *track.java*.

An illustration of the conversions between a variety of state files is shown in Figure 3.2. There are also other algorithms implemented. For example, the program *plot.java* reads in a *asa_out* format file, and extracts cost, accepted state, and generated state data along the searching process. Another program, *slot.java*, is used to average the searching routes from different random seeds. This routine divides the number of generated state into equal-widthed slots, and averages the all data points inside each slot. In Chapter 4, the implemented ASA algorithm will be used to analyze errors of the networks constructed for the target problems.

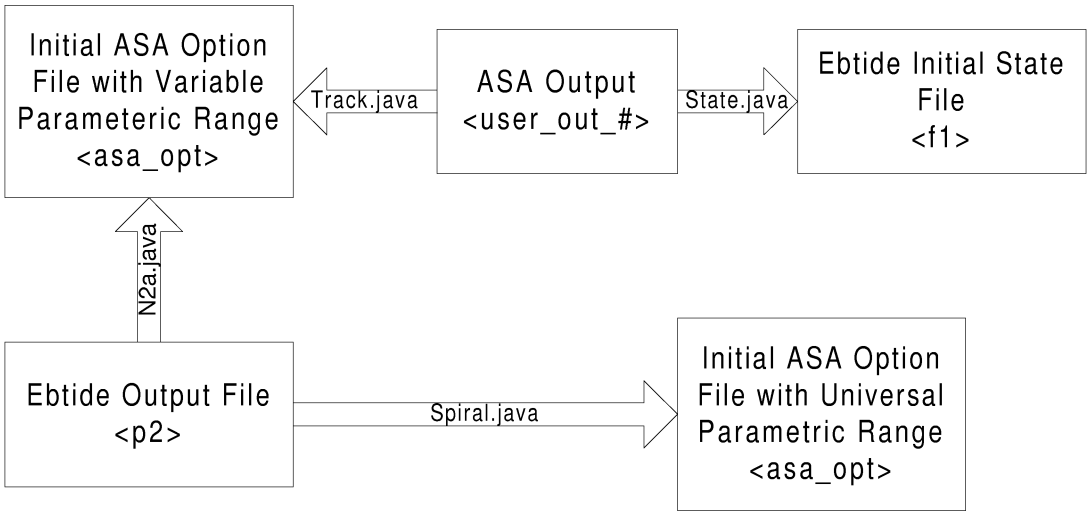

Figure 3.2: Diagram of Conversions between State Files

Chapter 4

Results

The approach to solve the problems with the ASA package was to try many possible combination of the options. After a number of attempts, the results would indicate certain trend of choices, or would give directions for further tuning the options in order to locate smaller errors.

Having obtained the minimum error, a series of runs was conducted by varying only the most important option, for example, the initial temperature of parameters while keeping other options unchanged. In addition, runs were conducted with different initial seed for the random number generator of the ASA package, under the same selection of options that gives the minimum.

Table 4.1: Program Options in *asa_opt* for Solving the Radiator Problem

Option	Value	Option	value
Limit_Acceptances	100000	Limit_Generated	999999
Cost_Precision	10^{-50}	Maximum_Cost_Repeat	100
Number_Cost_Samples	50	Temperature_Ratio_Scale	10^{-6}
Cost_Parameter_Scale_Ratio	1.0	Temperature_Anneal_Scale	10.0
Include_Integer_Parameters	FALSE	User_Initial_Parameters	FALSE
Sequential_Parameters	-1	Initial_Parameter_Temperature	10.0
Acceptance_Frequency_Modulus	100	Generated_Frequency_Modulus	10
Delta_X	0.001	*parameter_dimension	5

4.1 Results for the Radiator Problem

4.1.1 Conditions

The network constructed for solving the radiator problem is shown in Figure 2.3, yielding a total of 5 parameters for the cost function as the number of input, output, and hidden nodes are 2, 1, and 1, respectively. The selection of program options is shown in Table 4.1. The ranges of 5 parameters are arbitrarily chosen as $[-100, 10]$, $[-100, 20]$, $[-100, 100]$, $[-10, 10]$, and $[-10, 10]$, considering the optimal results from the back-propagation training are well within it, and all initial values of each parameter are set as 0.0.

4.1.2 Results

Using the back-propagation rule, the minimum prediction error, 0.011909, was reached by performing the learning steps shown in Table 4.2 with 30000 epochs. The table also gives the normalized error after each portion of training. The corresponding weight vectors $\mathbf{v}$ and $\mathbf{w}$ are $(15.0536, 38.854, -67.9959)$ and $(-0.00430794, -2.02565)$, respectively. The classification

Table 4.2: The Learning Process of Neural Networks for the Radiator Problem

Start Epoch	End Epoch	Hidden Layer Learning Rate	Output Layer Learning Rate	Normalized Error
1	20000	0.5	0.025	0.217320
20000	30000	0.001	10	0.011909

line drawn in Figure 2.6 is obtained by using $\mathbf{v} \cdot \mathbf{x} = 0$, where $\mathbf{x}$ is the input vector, and x_0 is set to 1.

The minimum error found by the ASA simulated annealing package is 0.004512939 when $\mathbf{v}$=(-18.53141, -47.8606, 84.06385) and $\mathbf{w}$=(-0.002072142, 2.011917). Although these weights are different from the ones obtained using back-propagation, the resultant classification plane are the same as shown in Figure 2.6. The process of the search using ASA is shown in Figure 4.1, and results show that the minimum error has been found after the algorithm generates more than 2000 states.

4.2 Results for the Spiral Problem

4.2.1 Conditions

The network constructed for solving the spiral problem is shown in Figure 2.8, yielding a total of 33 parameters for the cost function as the number of input, output, and hidden nodes are 2, 1, and 8, respectively. The selection of program options is shown in Table 4.3. The range of all parameters is chosen as -200 to 100 as the optimal results from the back-propagation training are well within it, and all initial values of each parameter are set as 0.0.

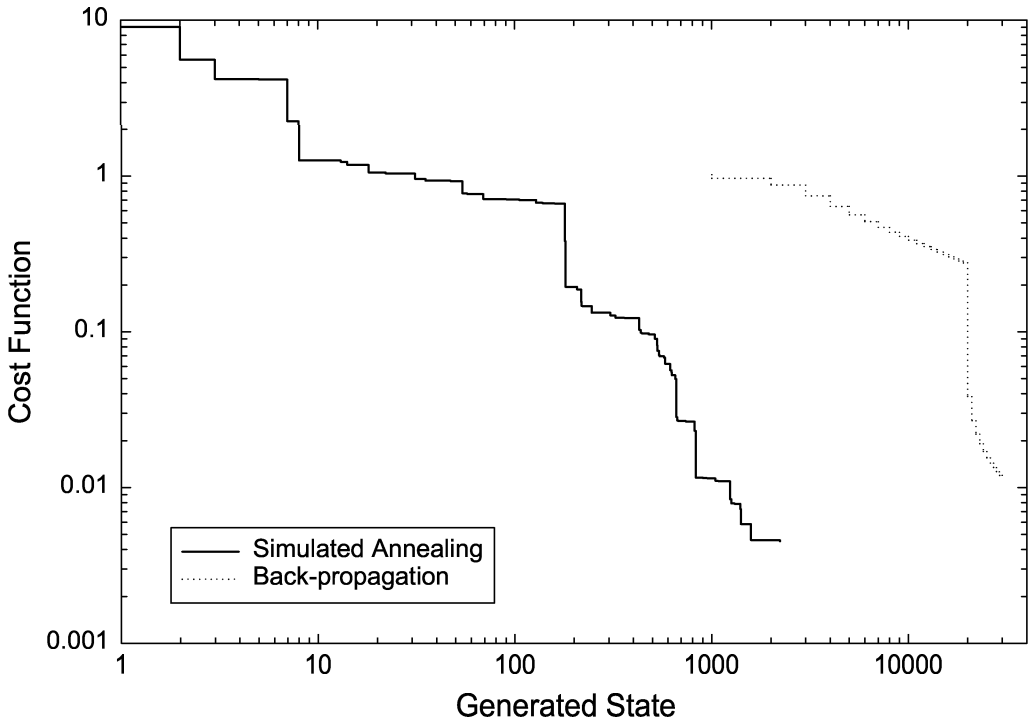

Figure 4.1: Searching Route for the Radiator Problem

Table 4.3: Program Options in *asa_opt* for Solving the Spiral Problem

Option	Value	Option	value
Limit_Acceptances	100000	Limit_Generated	2000000
Cost_Precision	10^{-50}	Maximum_Cost_Repeat	0
Number_Cost_Samples	30	Temperature_Ratio_Scale	10^{-10}
Cost_Parameter_Scale_Ratio	2.5	Temperature_Anneal_Scale	10^{8}
Include_Integer_Parameters	FALSE	User_Initial_Parameters	TRUE
Sequential_Parameters	-1	Initial_Parameter_Temperature	10^{-30}
Acceptance_Frequency_Modulus	1000	Generated_Frequency_Modulus	10
Initial Random Seed	696969	*parameter_dimension	33

Table 4.4: The Learning Process of Neural Networks for the Spiral Problem

Learning Step	Start Epoch	End Epoch	Hidden Layer Learning Rate	Output Layer Learning Rate	Normalized Error
1	0	5000	0.01	9	0.106245
2	5000	10000	0.001	18	0.021258
3	10000	15000	0.001	36	0.002503
4	15000	17500	0.0001	9	0.001836
5	17500	20000	0.0001	4.5	0.001783

4.2.2 Results

Using the back-propagation rule, the minimum prediction error, 0.001783, was reached by performing the learning steps shown in Table 4.4 with 20000 epochs. The table also gives the normalized error after each portion of training. The weight vectors yielding the minimum error

Table 4.5: Normalized Error for the Spiral Problem Using Different Initial Random Seeds by Back-Propagation

Initial Seed	Step 1	Step 2	Step 3	Step 4	Step 5
0	0.141293	0.332975	0.352216	0.136893	0.134228
111111	0.094488	0.16897	0.233462	0.112293	0.122002
222222	0.180575	0.259024	0.53778	0.538154	0.538112
333333	0.398334	0.310547	0.508095	0.50796	0.48616
444444	0.301612	0.315856	0.231731	0.227923	0.226423
555555	0.130145	0.126757	0.183104	0.114027	0.106825
666666	0.157928	0.201474	0.228211	0.129873	0.129743
777777	0.108987	0.504856	0.5332	0.508687	0.503907
888888	0.112501	0.168083	0.350085	0.137143	0.141566
999999	0.11548	0.174661	0.235491	0.218874	0.222174

are

$$\mathrm{v} = \begin{pmatrix} -46.9176 & 98.5965 & -14.741 \\ -14.687 & -9.49382 & 5.00181 \\ 50.3777 & 88.7164 & -0.933683 \\ 37.2328 & -6.52159 & -93.5482 \\ -45.2865 & 29.0718 & 63.8703 \\ -3.08139 & 79.716 & -65.7563 \\ -51.1781 & -21.3371 & -108.421 \\ 47.1278 & 36.9359 & 58.2973 \end{pmatrix}, \quad \mathrm{w} = \begin{pmatrix} 0.279811 \\ 0.999439 \\ 0.560316 \\ 1.00014 \\ 1.00051 \\ 1.0012 \\ -0.999146 \\ 0.99966 \\ 1 \end{pmatrix}.$$

For the purpose of testing statistical consistency, a number of initial seeds have been chosen for the random number generator described in section 2.2. The results, obtained using the same training steps as in Table 4.4, are shown in Table 4.5. The average normalized error over all

seeds using back-propagation is 0.261114.

The minimum error found by the ASA simulated annealing package is 0.072641, the corresponding accepted state and generated state are the 14117-*th* and the 1916830-*th*, respectively. The weight vectors giving the minimum for simulated annealing are

$$\mathrm{v} = \begin{pmatrix} 84.67782 & 54.79357 & 75.49387 \\ -59.75345 & 74.05624 & 84.38083 \\ -53.71343 & 41.91771 & 42.69367 \\ 0.5720288 & -28.33094 & 11.56554 \\ -52.17696 & -80.24953 & -57.29326 \\ -11.44992 & -9.050157 & 29.23013 \\ 9.509065 & -12.53225 & 21.97209 \\ 68.32424 & -7.782532 & -59.91836 \end{pmatrix}, \quad \mathrm{w} = \begin{pmatrix} 0.4444462 \\ 0.8405702 \\ 1.000876 \\ 1.604408 \\ 1.064329 \\ -1.008506 \\ -1.024244 \\ -1.067187 \\ -0.1203245 \end{pmatrix}.$$

Similarly, different initial random seeds can be chosen to test the consistency of the ASA algorithm, and the results are shown in Table 4.6. The average error is 0.270585. Figure 4.2 compares the training errors obtained using simulated annealing with those obtained using back-propagation, averaging over different seeds in both cases. The error vs. generated state curve for simulated annealing is obtained as described in section 3.2.3.

As the choice of initial temperature, T, for simulated annealing is crucial for the effective search, various initial temperatures have been tested, and the results are shown in Figure 4.3. More training results by the ASA package have been tabulated in Appendix A for the spiral problem.

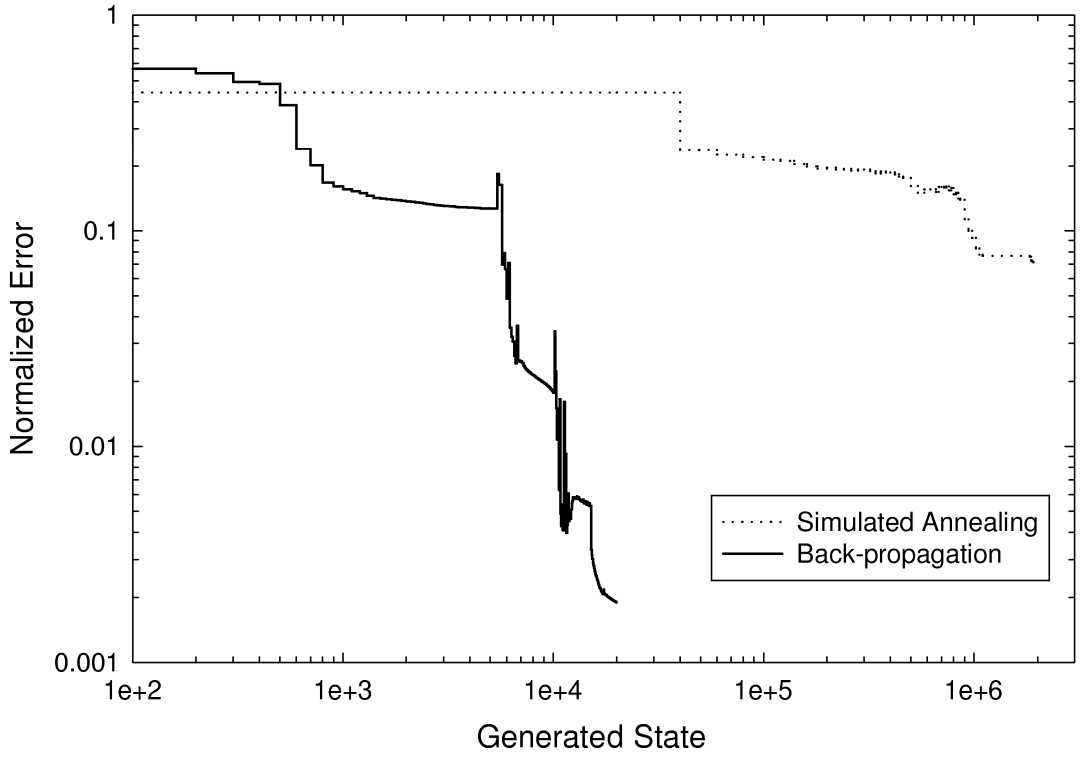

Figure 4.2: Error vs. Generated State, with Various Choice of Initial Random Seeds

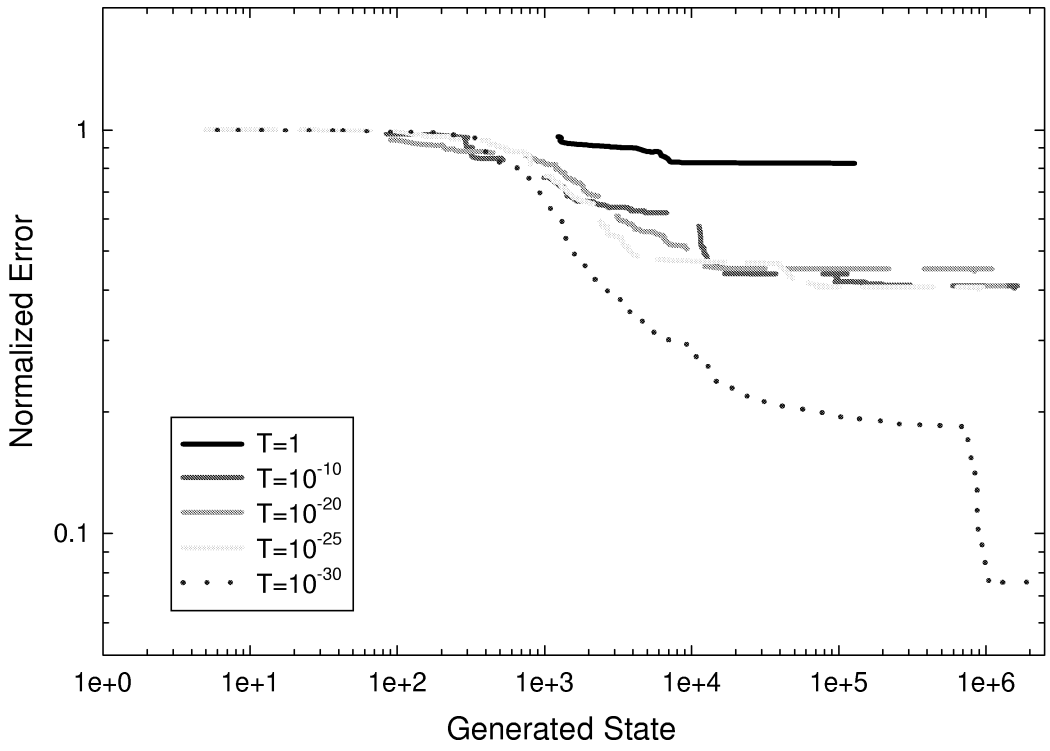

Figure 4.3: Error vs. Generated State, at Various Initial Parameter Temperatures

Table 4.6: Normalized Error for the Spiral Problem Using Different Initial Random Seeds by ASA

Initial Seed	Accepted State	Generated State	Normalized Error
0	2748	134570	0.2122665
111111	10974	1090080	0.2596658
222222	1959	147781	0.3058124
333333	10481	1882833	0.2795391
444444	13711	1525130	0.1353059
555555	6849	1888711	0.314253
666666	2855	148990	0.3614488
777777	15748	1465040	0.1890132
888888	2857	591843	0.3004423
999999	4506	1852552	0.3480993

4.3 Results for the Time Series Problem

4.3.1 Conditions

The network constructed for solving the time series prediction problem described in section 2.2.3, contains a total of 97 parameters for the cost function, since the number of input, output, and hidden nodes are 4, 1, and 16, respectively. The selection of program options is shown in Table 4.7. The ranges of the first 80 parameters are chosen from -60 to 90, and those of the next 17 parameters are from -2 to 3. The choice of the ranges are obtained from the back-propagation training data, and all initial values of each parameter are set as 0.0.

4.3.2 Results

Using the back-propagation rule, the minimum prediction error, 0.113213, was obtained by training as shown in Table 4.8 with 180000 epochs. The weight vectors $\mathbf{v}$ and $\mathbf{w}$ that give the

Table 4.7: Program Options in *asa_opt* for Solving the Time Series Prediction Problem

Option	Value	Option	value
Limit_Acceptances	1000000	Limit_Generated	1000000
Cost_Precision	10^{-50}	Maximum_Cost_Repeat	0
Number_Cost_Samples	30	Temperature_Ratio_Scale	0.1
Cost_Parameter_Scale_Ratio	2.5	Temperature_Anneal_Scale	10
Include_Integer_Parameters	FALSE	User_Initial_Parameters	FALSE
Sequential_Parameters	-1	Initial_Parameter_Temperature	10^{-31}
Acceptance_Frequency_Modulus	1000	Generated_Frequency_Modulus	10
Initial Random Seed	696969	*parameter_dimension	97

minimum error are

$$\mathrm{v} = \begin{pmatrix} -0.952957 & -16.4428 & -21.1504 & -19.4639 & 12.9193 \\ -14.1139 & -3.74407 & 64.5102 & -39.2345 & -50.0055 \\ 1.22682 & 27.4928 & 5.68114 & -24.9441 & -1.29383 \\ 4.06255 & 25.4381 & 15.7123 & -19.8415 & -32.006 \\ 5.83488 & -23.124 & -8.28563 & 22.982 & 10.6341 \\ 11.7906 & 6.67593 & 15.8109 & -56.4182 & -39.0922 \\ -2.03195 & -21.5789 & -2.51627 & 1.1009 & -36.8965 \\ 1.05458 & 20.1806 & 16.5118 & -2.77255 & 37.7577 \\ -6.40401 & 5.70832 & 27.9243 & 28.8256 & 4.73968 \\ -2.52614 & 15.8903 & -9.76329 & -13.0912 & 16.9695 \\ -13.2406 & 82.7416 & 11.4743 & 19.9098 & 13.6692 \\ -0.533864 & -0.0837724 & -2.22979 & 2.87875 & 6.91664 \\ 6.10691 & 11.7297 & 5.54728 & -49.1728 & -3.61223 \\ -0.896201 & 15.3482 & 14.7077 & 1.28529 & -0.213753 \\ -11.3206 & -30.1861 & -27.0799 & -16.6172 & -39.2277 \\ 5.03931 & 9.1686 & 16.2941 & 14.3579 & -1.0328 \end{pmatrix}, \quad \mathrm{w} = \begin{pmatrix} -0.919936 \\ -1.29411 \\ 1.60977 \\ 1.89828 \\ -1.39752 \\ 0.934614 \\ 2.05485 \\ -1.76577 \\ -1.12539 \\ 1.13095 \\ -1.44816 \\ -1.29357 \\ -0.864849 \\ -0.930569 \\ -1.34223 \\ 2.5527 \\ 1.49516 \end{pmatrix}.$$

Table 4.8: The Learning Process of Neural Networks for the Time Series Prediction Problem

Start Epoch	End Epoch	Hidden Layer Learning Rate	Output Layer Learning Rate	Normalized Error
0	180000	0.1	1.0	0.113213

A number of initial seeds have been chosen for the random number generator as described in section 2.2. The results obtained for these seeds with training as in Table 4.8, are shown in Table 4.9, and the average normalized error using back-propagation is 0.148800.

The minimum error located by the ASA package is 0.2499432, and the corresponding accepted state and generated state are the 132187-*th* and the 3336034-*th*, respectively. The weight

vectors giving the minimum are

$$\mathrm{v} = \begin{pmatrix} -27.49038 & 83.90247 & -11.77074 & -55.13692 & -47.44127 \\ -13.61337 & -43.38077 & -0.4851812 & 61.88745 & 62.06438 \\ -1.650036 & 83.46525 & -24.17691 & 4.140091 & -49.47931 \\ 21.98708 & 80.09414 & 30.46586 & 42.4696 & 76.66342 \\ 12.74138 & -3.906566 & 36.39858 & -58.6336 & -59.23762 \\ 3.197878 & 43.53093 & 2.213536 & 40.96112 & -27.25932 \\ 5.228675 & 27.86285 & 22.92234 & -1.295725 & 54.69305 \\ -14.41241 & 76.60146 & 87.09359 & 5.733913 & 40.5343 \\ 0.4092121 & 0.8601366 & 0.5724198 & 0.1880955 & -1.062659 \\ 1.162809 & 0.5216952 & 0.6678509 & 0.1987488 & 75.01283 \\ 79.57493 & 66.99852 & 51.38487 & 81.80978 & -33.77488 \\ 37.73038 & -31.84893 & -55.46933 & 1.241327 & 10.206 \\ -22.41386 & 50.59685 & -49.46794 & 2.311856 & 27.65929 \\ -39.27847 & 47.03806 & -54.72295 & 46.68399 & 70.91087 \\ 36.55967 & 59.77812 & -58.91814 & -38.96403 & 22.34014 \\ -21.59117 & -5.429523 & -2.333327 & -46.88641 & 41.58648 \end{pmatrix}, \quad \mathrm{w} = \begin{pmatrix} -1.755945 \\ 1.305737 \\ 1.898021 \\ -1.079981 \\ 1.13089 \\ 2.822487 \\ 0.3133936 \\ 2.146596 \\ 1.645956 \\ 2.071676 \\ 2.699167 \\ -0.2576602 \\ 1.065784 \\ -1.445966 \\ 1.768835 \\ -1.370405 \\ -1.69499 \end{pmatrix}.$$

Similarly, different initial random seeds were chosen to test the statistical consistency of the ASA algorithm, and the results shown in Table 4.10. The average error is 0.345336. Figure 4.4 compares the average training error obtained using back-propagation and simulated annealing. The error vs. generated state curve for simulated annealing is obtained as described in section 3.2.3.

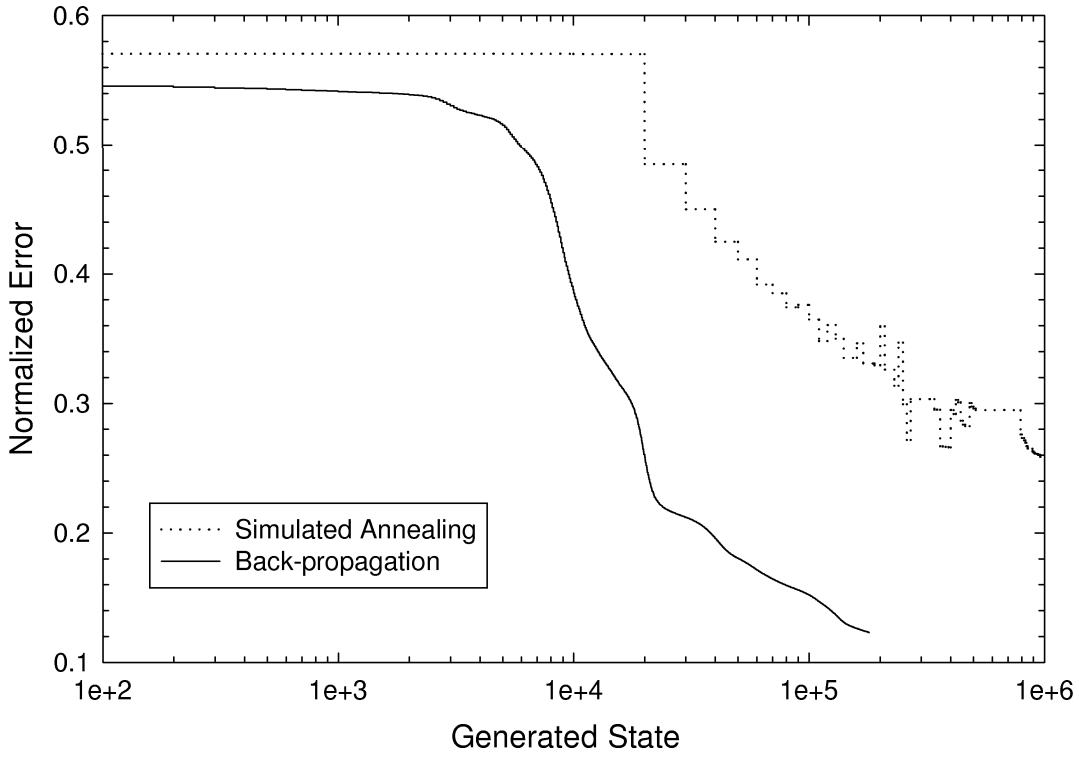

Figure 4.4: Error vs. Generated State, with Various Choice of Initial Random Seeds

Table 4.9: Normalized Error for the Time Series Prediction Problem Using Different Initial Random Seeds by Back-Propagation

Initial Seed	Normalized Error
0	0.155916
111111	0.137243
222222	0.141591
333333	0.175674
444444	0.139120
555555	0.122200
666666	0.156364
777777	0.137673
888888	0.173097
999999	0.149123

As the choice of initial temperatures for simulated annealing are crucial for the effective search, various initial temperatures have been tested, and the results are shown in Figure 4.5. More training results by the ASA package have been tabulated in Appendix B for the time series prediction problem.

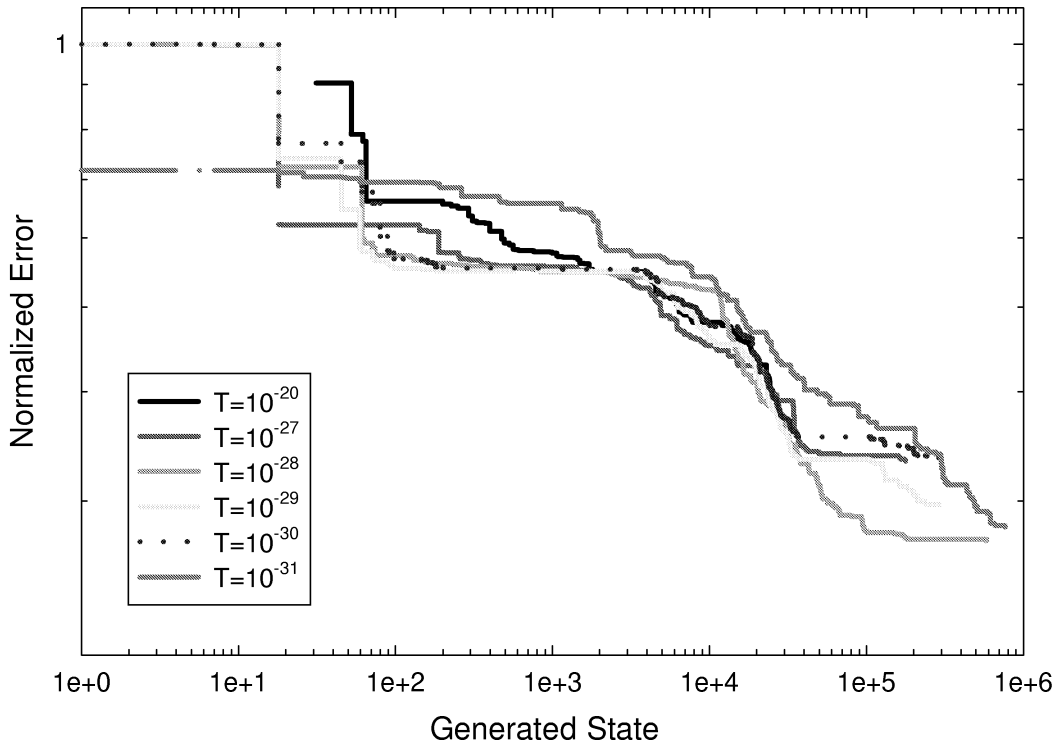

Figure 4.5: Error vs. Generated State, at Various Initial Parameter Temperatures

Table 4.10: Normalized Error for the Time Series Prediction Problem Using Different Initial Random Seeds by ASA

Initial Seed	Accepted State	Generated State	Normalized Error
0	7808	171739	0.311563
111111	36242	1090484	0.2917925
222222	74986	1420897	0.4009745
333333	7056	151851	0.3590085
444444	2322	25289	0.3761782
555555	8015	182922	0.3299827
666666	9832	153838	0.3852046
777777	12156	383232	0.2653022
888888	11186	228410	0.3907496
999999	5418	150771	0.3426085

Chapter 5

Conclusions

In Chapter 4, three target problems were investigated, i.e., the radiator problem with 5 weights, the spiral problem with 33 weights, and the time series prediction problem with 97 weights. Resultant minimal errors of the neural networks obtained using both back-propagation and simulated annealing algorithms were presented for each problem.

In the case of the radiator problem, a smaller error was found by the ASA package than by back-propagation training. However, in the problems with larger dimension in the parameter space, such as the spiral and the time series prediction problems, the ASA package was unable to locate errors as small as the back-propagation method in the computation time available. This indicates that the ASA algorithm does not yield more accurate results than back-propagation when a parameter space of large dimension needs to be traversed. In the case of 33–dimensional space, as in the spiral problem, the ASA algorithm gives a larger error, 0.270585, compared to 0.261114 from the back-propagation analysis; and in the case of 97–dimensional space, as in

the time series prediction problem, the resulting error from ASA, 0.345336, is 2.3 times that from back-propagation, 0.1488.

The ASA algorithm also consumes longer computing time than back propagation. In general, the time taken by an ASA search is approximately 2 to 10 times of that of back-propagation. The addition of nodes in the hidden layer increases the dimensionality of the search space, greatly increasing the searching time. For example, the time taken to locate the minimum error in the spiral problem is around one third that of the time series prediction problem, which is normally 24 to 48 hours. Therefore, the ASA algorithm is more effective in relatively small dimensions of parameter space, e.g., less than 32 dimensions, as demonstrated in the radiator problem and the spiral problem. However, this does not mean that the ASA algorithm is incapable of finding the global optimal point, given proper setting of the options and long enough training time.

In the two problems with large parameter space dimension, results show that only 3 program options are crucial to the final outcome. Most importantly, the initial parameter temperature must be set very low, below 10^{-30} for both cases. It was found that low initial temperature leads to a search with small steps, i.e., more detailed search. In contrast, the search with high initial temperatures would normally ignore the details of the abruptly rising hills or falling valleys of the parameter space, which is seemed to be the case for both the problems.

The temperature ratio scale and the temperature annealing scale are the next two important factors, and they control the cooling schedule. They appear to be the next two options that need to be tuned after the proper initial parameter temperature is found. It also appears that the

product of the two options should be in the range from 0.1 to 10, and under such conditions, the ASA algorithm would give an efficient search for the problems at hand.

For future work, simulated annealing can be used as an alternative to back-propagation for minimizing errors in neural networks, and it is especially effective when only a moderate number of parameters are considered, for example, less than 32. Meanwhile, the simulated annealing algorithm can also be applied to neural networks with multi-modal error functions, in which case the back-propagation search can be easily trapped at a local minimum. Simulated annealing can theoretically guarantee the location of a global minimum error given enough search time and computational power. In addition, an advantage of the simulated annealing algorithm over other gradient descent based methods is that it can search over a discrete parameter space.

Bibliography

Bibliography

[1] *CS 522: Cybernetics*, B. Whitehead Ed., Fall 1999; and class notes used in CS 522 Cybernetics class given at the University of Tennessee Space Institute, Fall 1999.

[2] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, **Addison-Wesley Publishing Company**, Chapter 5, 1991; and references therein.

[3] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagation errors", *Nature* **323** pp. 533–536, 1986.

[4] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks for robot control", *Neural networks* **2(5)** pp. 359–366, 1989.

[5] K. Funahashi, "On the approximate realization of continuous mappings by neural networks", *Neural Networks* **2(3)** pp. 192–193, 1989.

[6] G. Cybenco, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems* **2(4)** pp. 303–314, 1989.

[7] E. Hartman, J. Keeler, and J. Kowalski, "Layered neural networks with Gaussian hidden units as universal approximations", *Neural Computation* **2(2)** pp. 210–215, 1990.

[8] K. Lang, and M. Witbrock, "Learning to tell two spirals apart", *Proc. of the 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski Eds., **Morgan Kaufmann Publishers, Inc.** pp. 52–59, 1989.

[9] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems", *Science* **197** pp. 287–289, 1977.

[10] B. Whitehead, and T. Choate, "Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction", *IEEE Trans. on Neural Networks* **7(4)** pp. 869–880, 1996.

[11] E. Aarts and J. Korst, *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*, **Wiley–Interscience Publishing Company**, Chapter 1–2, 1989; and references therein.

[12] L. Ingber, "Adaptive simmulated annealing(ASA): leassons learned", *J. Control and Cybernetics* **25(1)**, pp. 33–54, 1996.

[13] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing", *Science* **220**, pp. 671–680, 1983.

[14] N. Metroplis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines", *J. Chem. Phys.* **21** pp. 1087–1092, 1953.

[15] S. Geman, and D. Geman, “Stochastic relaxation, Gibbs distribution and the Bayesian restoration in images”, *IEEE Trans. Patt. Anal. Mac. Int.* **6(6)** pp. 721–741, 1953.

[16] H. Szu, and R. Hartley, “Fast simulated annealing”, *Phys. Lett.* **A 122(3–4)** pp. 157–162, 1987.

[17] K. Binder, D. Stauffer, “A simple introduction to Monte Carlo simulation and some specialized topics”, *Applications of the Monte Carlo methods in statistical physics*, K. Binder Ed., **Springer-Verlag, Berlin**, pp. 1–36, 1985; and references therein.

[18] L. Ingber, “Options for large space”, *Adaptive Simulated Annealing* Notes, Version 17.19.

Appendix

Appendix A

Results for Spiral Problem

Table A.1: Program Options in *asa_opt* for Solving the Spiral Problem

Option	Value	Option	value
Limit_Acceptances	1×10^{6}	Limit_Generated	1×10^{7}
Cost_Precision	10^{-50}	Maximum_Cost_Repeat	0
Include_Integer_Parameters	FALSE	User_Initial_Parameters	TRUE
Sequential_Parameters	-1	Accepted_To_Generated_Ratio	10^{-8}
Acceptance_Frequency_Modulus	1×10^{3}	Generated_Frequency_Modulus	10
Initial Random Seed	696969	*parameter_dimension	33

Table A.2: Results for the Spiral Problem, under Solaris O.S. 5.7

Number Cost Samples	Temperature Ratio Scale	Cost Parameter Ratio Scale	Temperature Anneal Scale	Initial Parameter Temperature	Normalized Error	Corresponding Generated State
30	1×10^{-11}	2.5	1×10^{9}	1×10^{-5}	0.444613	72171
30	1×10^{-10}	2.5	1×10^{9}	1×10^{-5}	0.625647	67081
30	1×10^{-12}	2.5	1×10^{9}	1×10^{-5}	0.5768676	70229
30	1×10^{-12}	2.5	1×10^{10}	1×10^{-5}	0.3633625	898810
30	1×10^{-11}	3	1×10^{9}	1×10^{-5}	0.4360099	3309582
30	1×10^{-13}	3	1×10^{11}	1×10^{-5}	0.5015484	31713
30	1×10^{-14}	3	1×10^{12}	1×10^{-5}	0.5227651	89025
30	1×10^{-10}	3	1×10^{8}	1×10^{-5}	0.494013	63863
30	1×10^{-12}	3	1×10^{11}	1×10^{-5}	0.5223265	61394
30	1×10^{-9}	3	1×10^{7}	1×10^{-5}	0.6054162	58409
30	1×10^{-13}	3	1×10^{15}	1×10^{-5}	0.6774632	30864
30	1×10^{-13}	3	1×10^{13}	1×10^{-5}	0.5732937	99912
30	1×10^{-11}	3	1×10^{10}	1×10^{-5}	0.600413	54221
30	1×10^{-11}	3	1×10^{8}	1×10^{-5}	0.5350931	75158
100	1×10^{-11}	3	1×10^{9}	1×10^{-5}	0.8529355	12500
100	0.1	3	100	1×10^{-5}	0.897058	81960
100	1×10^{-5}	3	1×10^{4}	1×10^{-5}	0.8543712	80197
30	1×10^{-11}	2.5	1×10^{9}	8×10^{-6}	0.3758946	62427
30	1×10^{-11}	2.5	1×10^{9}	7×10^{-6}	0.3705327	32205
30	1×10^{-11}	2.5	1×10^{9}	6×10^{-6}	0.4556816	47024
30	1×10^{-11}	2.5	1×10^{9}	7×10^{-6}	0.3375088	1683408
30	1×10^{-11}	2.5	1.1×10^{9}	7×10^{-6}	0.585448	97083
30	1×10^{-11}	2.5	9×10^{8}	7×10^{-6}	0.403476	81583
50	1×10^{-11}	2.5	1×10^{8}	1×10^{-5}	0.697576	46194
50	1×10^{-11}	2.5	1×10^{7}	1×10^{-5}	0.5024043	91546
50	1×10^{-11}	2.5	1×10^{6}	1×10^{-5}	0.4299886	84621
50	1×10^{-11}	2.5	1×10^{5}	1×10^{-5}	0.4274408	23600
50	1×10^{-11}	2.5	1×10^{4}	1×10^{-5}	0.4964668	49865
50	1×10^{-12}	2.5	1×10^{5}	1×10^{-5}	0.4795088	97888
50	1×10^{-11}	2.5	1×10^{6}	1×10^{-5}	0.4219956	101522
50	1×10^{-11}	2.5	1×10^{6}	2×10^{-5}	0.6232623	70209
50	1×10^{-10}	2.5	1×10^{6}	1×10^{-6}	0.9819123	16160

Table A.3: Results for the Spiral Problem, under Solaris O.S. 5.7, continued from Table A.2

Number Cost Samples	Temperature Ratio Scale	Cost Parameter Ratio Scale	Temperature Anneal Scale	Initial Parameter Temperature	Normalized Error	Corresponding Generated State
50	1×10^{-10}	2.5	15	1×10^{-6}	0.5347028	37650
20	1×10^{-8}	2.5	1×10^{8}	1×10^{-5}	0.5769895	90073
20	1×10^{-8}	2.5	1×10^{9}	1×10^{-5}	0.6572871	17243
20	1×10^{-8}	2.5	1×10^{7}	1×10^{-5}	0.8516324	94727
20	1×10^{-8}	2.5	1×10^{10}	1×10^{-5}	0.9506295	56730
20	1×10^{-7}	2.5	1×10^{7}	1×10^{-5}	0.740145	33425
20	1×10^{-7}	2.5	1×10^{8}	1×10^{-5}	0.7964698	89537
20	1×10^{-7}	2.5	1×10^{6}	1×10^{-5}	0.7486402	65018
50	1×10^{-11}	2.5	1×10^{5}	1×10^{-5}	0.5299483	9146
50	1×10^{-8}	2.5	1×10^{9}	1×10^{-5}	0.6953293	9436
50	1×10^{-8}	2.5	1×10^{7}	1×10^{-5}	0.8198425	75106
50	1×10^{-8}	2.5	1×10^{10}	1×10^{-5}	0.6828337	67399
50	1×10^{-8}	2.5	1×10^{11}	1×10^{-5}	0.7237347	12700
50	1×10^{-6}	2.5	1×10^{8}	1×10^{-5}	0.7351596	93243
50	1×10^{-6}	2.5	1×10^{5}	1×10^{-5}	0.7688819	21309
10	1×10^{-11}	2.5	1×10^{10}	1×10^{-5}	1	0
10	1×10^{-11}	2.5	1×10^{3}	1×10^{-5}	0.5785147	88345
10	1×10^{-11}	2.5	1×10^{4}	1×10^{-5}	0.7123952	24298
10	1×10^{-11}	2.5	100	1×10^{-5}	0.6800098	99710
10	1×10^{-11}	2.5	1×10^{3}	1×10^{-5}	0.5785147	88341
10	1×10^{-12}	2.5	1×10^{6}	1×10^{-5}	0.6691641	51722
10	1×10^{-8}	2.5	1×10^{8}	1×10^{-5}	0.8578451	2420
10	1×10^{-8}	2.5	1×10^{10}	1×10^{-5}	0.8107116	34045
10	1×10^{-8}	2.5	1×10^{12}	1×10^{-5}	1	0
10	0.01	2.5	1×10^{4}	1×10^{-5}	0.9176276	46535
10	0.01	2.5	1×10^{6}	1×10^{-5}	0.9753093	2420
10	0.01	2.5	100	1×10^{-5}	1	0
10	0.01	2.5	1×10^{3}	1×10^{-5}	1	0
10	0.01	2.5	1×10^{10}	1×10^{-5}	0.9834828	14388
10	0.01	2.5	1×10^{2}	1×10^{-5}	0.9952903	3443
30	1×10^{-11}	2.5	1×10^{9}	1×10^{-10}	0.4492905	99553
30	1×10^{-11}	2.5	1×10^{9}	1×10^{-20}	0.4234414	125233

Table A.4: Results for the Spiral Problem, under Solaris O.S. 5.7, continued from Table A.3

Number Cost Samples	Temperature Ratio Scale	Cost Parameter Ratio Scale	Temperature Anneal Scale	Initial Parameter Temperature	Normalized Error	Corresponding Generated State
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-30}	0.0835752	999944
30	1×10^{-9}	2.5	1×10^{7}	1×10^{-30}	0.2037759	
30	1×10^{-10}	2.5	1×10^{9}	1×10^{-30}	0.5052053	2991613
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-35}	0.5203233	2556313
30	1×10^{-10}	2.5	1×10^{8}	9×10^{-31}	0.2137464	1911350
30	1×10^{-10}	2.5	1×10^{8}	1.5×10^{-30}	0.3446315	1537040
30	1×10^{-11}	2.5	1×10^{9}	1×10^{-30}	0.4025038	1999316
30	1×10^{-11}	2.5	1×10^{9}	1×10^{-30}	0.3969867	9999351
30	1×10^{-8}	2.5	1×10^{6}	1×10^{-30}	0.1739831	9820923
50	1×10^{-5}	2.5	1×10^{3}	1×10^{-30}	0.4070411	341810
30	1×10^{-5}	2.5	1×10^{3}	1×10^{-30}	0.3379628	37130
30	1×10^{-6}	2.5	1×10^{4}	1×10^{-30}	0.2505771	950449
10	1×10^{-6}	2.5	1×10^{4}	1×10^{-30}	0.6333301	38990
30	0.01	2.5	1	1×10^{-30}	0.247285	3994826
30	0.01	2.5	1	1×10^{-20}	0.3572507	3438556
40	0.01	2.5	1	1×10^{-30}	0.2986421	
40	0.01	2.5	1	1×10^{-20}	0.46042	261311
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-1}	0.8242204	128040
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-5}	0.426159	410781
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-10}	0.4046287	1563180
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-20}	0.4095569	1299138
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-25}	0.4021808	899491
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-26}	0.4485726	1992870
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-27}	0.434362	178880
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-28}	0.3164281	1237563
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-29}	0.3264368	81890
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-30}	0.0715647	1916830
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-31}	0.4156436	1210449
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-32}	0.2387071	1999460
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-33}	0.3422933	84901
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-34}	0.2623347	1746603
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-35}	0.5206659	1540920
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-40}	0.3342534	644722

Appendix B

Results for Time Series Prediction Problem

Table B.1: Program Options in *asa_opt* for Solving the Time Series Prediction Problem

Option	Value	Option	value
Limit_Acceptances	1×10^{6}	Limit_Generated	5000000
Cost_Precision	10^{-50}	Maximum_Cost_Repeat	0
Include_Integer_Parameters	FALSE	User_Initial_Parameters	TRUE
Sequential_Parameters	-1	Accepted_To_Generated_Ratio	10^{-8}
Acceptance_Frequency_Modulus	1×10^{3}	Generated_Frequency_Modulus	10
Initial Random Seed	696969	*parameter_dimension	97

Table B.2: Results for the Time Series Predition Problem, under Solaris O.S. 5.6

Number Cost Samples	Temperature Ratio Scale	Cost Parameter Ratio Scale	Temperature Anneal Scale	Initial Parameter Temperature	Normalized Error	Corresponding Generated State
100	1×10^{-9}	3	1×10^{9}	1×10^{-5}	0.5575335	20946
100	1×10^{-8}	3	1×10^{9}	1×10^{-5}	1.469954	14090
100	1×10^{-8}	3	1×10^{8}	1×10^{-5}	0.5728139	9340
100	1×10^{-10}	3	1×10^{9}	1×10^{-5}	1.479802	26548
100	1×10^{-9}	3	1×10^{10}	1×10^{-5}	0.556686	21092
100	1×10^{-9}	3	1×10^{11}	1×10^{-5}	0.5713645	9024
100	1×10^{-9}	3	1×10^{12}	1×10^{-5}	0.8652808	10319
100	1×10^{-10}	3	1×10^{11}	1×10^{-5}	1.498423	3881
90	1×10^{-9}	3	1×10^{10}	1×10^{-5}	0.5873134	11480
50	1×10^{-9}	3	1×10^{10}	1×10^{-5}	0.6525968	24134
30	1×10^{-11}	3	1×10^{14}	1×10^{-5}	0.6035243	11135
30	1×10^{-11}	3	1×10^{13}	1×10^{-5}	0.5758992	28904
30	1×10^{-9}	3	1×10^{10}	1×10^{-5}	0.6026971	119700
30	1×10^{-9}	3	1×10^{9}	2×10^{-5}	0.6037148	2480
30	1×10^{-9}	3	1×10^{9}	5×10^{-6}	0.6043706	27124
30	1×10^{-9}	3	1×10^{10}	2×10^{-5}	0.5930593	8610
30	1×10^{-9}	3	1×10^{10}	1×10^{-4}	0.5902323	8613
30	1×10^{-9}	3	1×10^{10}	1×10^{-3}	0.9279706	6918
30	1×10^{-1}	3	1×10^{10}	1×10^{-4}	0.8992819	7400
30	1×10^{-9}	3	1×10^{11}	1×10^{-4}	0.890934	7738
30	1×10^{-9}	3	1×10^{12}	1×10^{-4}	0.6226148	2825
30	1×10^{-9}	3	1×10^{12}	1×10^{-4}	0.6151202	57373
200	1×10^{-9}	3	1×10^{10}	1×10^{-5}	0.6246414	10562
200	1×10^{-10}	3	1×10^{11}	1×10^{-5}	0.7164145	10230
200	1×10^{-8}	3	1×10^{9}	1×10^{-5}	0.9676952	19747
30	1×10^{-11}	3	1×10^{12}	1×10^{-5}	0.5497964	20751
30	1×10^{-12}	3	1×10^{13}	1×10^{-5}	0.553421	7460
30	1×10^{-12}	3	1×10^{13}	9×10^{-6}	0.5782821	4288
30	1×10^{-12}	3	1×10^{13}	1.1×10^{-5}	0.5685024	9449
30	1×10^{-11}	3	1×10^{12}	9×10^{-6}	0.8310312	17223
10	1×10^{-11}	3	1×10^{12}	1×10^{-5}	0.5299483	9146
1	1×10^{-11}	3	1×10^{12}	1×10^{-5}	0.5644109	6530
10	1×10^{-8}	3	1×10^{9}	1×10^{-5}	0.5418674	17201
10	1×10^{-7}	3	1×10^{8}	1×10^{-5}	0.78385	2135

Table B.3: Results for the Time Series Predition Problem, under Solaris O.S. 5.6, continued from Table B.2

Number Cost Samples	Temperature Ratio Scale	Cost Parameter Ratio Scale	Temperature Anneal Scale	Initial Parameter Temperature	Normalized Error	Corresponding Generated State
1	1×10^{-8}	3	1×10^{9}	1×10^{-5}	0.6039516	2860
10	1×10^{-9}	3	1×10^{10}	1×10^{-5}	0.5501372	4092
10	1×10^{-12}	3	1×10^{13}	1×10^{-5}	0.6491504	3458
10	1×10^{-10}	3	1×10^{11}	1×10^{-5}	0.5506838	6568
10	1×10^{-11}	3	1×10^{12}	9×10^{-6}	0.5625806	29833
10	1×10^{-11}	3	1×10^{12}	1.1×10^{-5}	0.5322258	21506
15	1×10^{-11}	3	1×10^{12}	1.1×10^{-5}	0.6507482	15961
15	1×10^{-11}	3	1×10^{12}	1×10^{-5}	0.6403359	24230
15	1×10^{-11}	3	1×10^{12}	1×10^{-6}	0.5741667	3728
15	1×10^{-11}	3	1×10^{12}	1×10^{-9}	0.5370556	21270
15	1×10^{-11}	3	1×10^{12}	1×10^{-20}	0.5296859	10356
15	1×10^{-11}	3	1×10^{12}	1×10^{-20}	0.4654175	58918
15	1×10^{-11}	3	1×10^{12}	1×10^{-30}	0.3248053	76000
15	1×10^{-10}	3	1×10^{11}	1×10^{-30}	0.2942582	299958
15	1×10^{-10}	3	1×10^{11}	1×10^{-30}	0.2934597	313560
15	1×10^{-9}	3	1×10^{10}	1×10^{-30}	0.372256	N/A
10	1×10^{-11}	3	1×10^{12}	1×10^{-30}	0.4444849	165080
30	1×10^{-9}	3	1×10^{10}	1×10^{-30}	0.3382412	123783
10	1×10^{-11}	3	1×10^{12}	1×10^{-10}	0.5101448	63631
10	1×10^{-11}	3	1×10^{12}	1×10^{-40}	0.4003304	99903
15	1×10^{-10}	3	1×10^{11}	1×10^{-40}	0.2761454	99956
15	1×10^{-10}	3	1×10^{11}	1×10^{-40}	0.2658189	467301
15	1×10^{-10}	3	1×10^{11}	1×10^{-45}	N/A	0
15	1×10^{-5}	3	1×10^{6}	1×10^{-30}	0.3595894	45683
15	1×10^{-5}	3	1×10^{7}	1×10^{-30}	0.3998911	893767
15	1×10^{-8}	3	1×10^{9}	1×10^{-20}	0.4869963	46801
15	1×10^{-8}	3	1×10^{9}	1×10^{-30}	0.33978	156560
30	1×10^{-8}	3	1×10^{9}	1×10^{-30}	0.2961404	76050
50	1×10^{-8}	3	1×10^{9}	1×10^{-30}	0.4374687	1997291
30	1×10^{-10}	3	1×10^{11}	1×10^{-40}	0.4395727	194400
30	0.1	3	100	1×10^{-30}	0.3320647	194303
30	0.1	3	100	1×10^{-20}	0.4342545	42440
30	0.1	3	100	1×10^{-40}	0.2859032	1351311
40	0.1	3	100	1×10^{-40}	0.3024414	1031399

Table B.4: Results for the Time Series Predition Problem, under Solaris O.S. 5.6, continued from Table B.3

Number Cost Samples	Temperature Ratio Scale	Cost Parameter Ratio Scale	Temperature Anneal Scale	Initial Parameter Temperature	Normalized Error	Corresponding Generated State
1×10^{3}	0.1	3	100	1×10^{-40}	0.3297893	612525
1×10^{3}	0.1	3	100	1×10^{-30}	1.000041	0
30	1×10^{-10}	2.5	1×10^{8}	1×10^{-30}	0.4198583	76514
30	1	2.5	1	1×10^{-30}	0.76596141	813379
30	0.1	2.5	1	1×10^{-30}	0.3133045	516641
30	0.1	2.5	10	1×10^{-30}	0.2698661	763878
30	0.1	2.5	100	1×10^{-30}	0.2968669	1312263
30	0.01	2.5	100	1×10^{-30}	0.3762218	1036514
30	0.01	2.5	10	1×10^{-30}	0.5050387	15156
30	0.01	2.5	1×10^{5}	1×10^{-30}	0.4330726	43829
30	0.01	2.5	100	1×10^{-30}	0.3762218	1036514
30	1×10^{-3}	2.5	100	1×10^{-30}	0.3492723	48590
30	1×10^{-4}	2.5	100	1×10^{-30}	0.3639588	32250
15	1×10^{-10}	3	1×10^{11}	1×10^{-10}	2.074897	237239
15	1×10^{-10}	3	1×10^{11}	1×10^{-39}	0.43298	463894
15	1×10^{-10}	3	1×10^{11}	1×10^{-42}	0.9331574	39
15	1×10^{-10}	3	1×10^{11}	1×10^{-38}	0.3641431	236784
15	1×10^{-10}	3	1×10^{11}	1×10^{-36}	0.4040927	129306
30	0.1	2.5	10	1	1.001633	521
30	0.1	2.5	10	1×10^{-5}	0.9868699	1027
30	0.1	2.5	10	1×10^{-10}	0.5437543	58887
30	0.1	2.5	10	1×10^{-15}	0.6909895	36459
30	0.1	2.5	10	1×10^{-20}	0.3944421	25602
30	0.1	2.5	10	1×10^{-26}	0.2979636	249997
30	0.1	2.5	10	1×10^{-27}	0.3330565	177216
30	0.1	2.5	10	1×10^{-28}	0.2701167	583162
30	0.1	2.5	10	1×10^{-29}	0.297153	290936
30	0.1	2.5	10	1×10^{-30}	0.2698661	763878
30	0.1	2.5	10	1×10^{-31}	0.2499432	3336034
30	0.1	2.5	10	1×10^{-32}	0.2770364	751914
30	0.1	2.5	10	1×10^{-33}	0.2818749	749746
30	0.1	2.5	10	1×10^{-34}	0.308731	750790
30	0.1	2.5	10	1×10^{-35}	0.2549542	875540
30	0.1	2.5	10	1×10^{-36}	0.3623526	215652
30	0.1	2.5	10	1×10^{-40}	0.2826464	333811

Table B.5: Results for the Time Series Predition Problem, under Solaris O.S. 5.7

Number Cost Samples	Temperature Ratio Scale	Cost Parameter Ratio Scale	Temperature Anneal Scale	Initial Parameter Temperature	Normalized Error	Corresponding Generated State
15	1×10^{-10}	3	1×10^{11}	1×10^{-40}	0.3689598	893277
30	1×10^{-10}	3	1×10^{11}	1×10^{-40}	0.3567228	224283
50	1×10^{-10}	3	1×10^{11}	1×10^{-40}	0.361607	197720
30	1	3	100	1×10^{-20}	0.6362035	636474
100	1	3	100	1×10^{-30}	0.8129627	859586
100	1×10^{-10}	3	1×10^{11}	1×10^{-20}	0.4728448	1259841
100	1×10^{-10}	3	1×10^{11}	1×10^{-30}	0.6115052	12270
100	1×10^{-10}	3	1×10^{11}	1×10^{-10}	0.5447494	7541
15	1×10^{-10}	3	1×10^{11}	1×10^{-41}	0.2906213	116011
15	1×10^{-10}	3	1×10^{11}	1×10^{-37}	0.3321331	836763
15	1×10^{-10}	3	1×10^{11}	1×10^{-35}	0.3108865	99580
30	0.1	2.5	10	1×10^{-30}	0.3821284	193780
30	0.1	2.5	10	1×10^{-32}	0.2722752	394589
30	0.1	2.5	10	1×10^{-31}	0.294269	120728

Vita

Yuxing Sun was born in Huhhot, Inner Mongolia of the People's Republic of China on December 17, 1970. He attended elementary school in Huhhot, and graduated from Huhhot No. 2 Middle School in July, 1988. He entered Peking University, at Beijing, China in September 1988, and received his Bachelor of Science Degree in Physics in July, 1992. He continued with his education in the Physics Department of Peking University, and received his Master of Science degree in Physics in July of 1995. In August, 1995, he came to the United States of America, and entered the Doctor's program in Physics at The University of Tennessee Space Institute, and officially received his Doctor of Philosophy Degree in Physics in May, 2000. In May of 1999, he also entered the Master's program in Computer Science, and received his Master of Science Degree in Computer Science in May, 2001.