6-16-2016

Synaptic Annealing: Anisotropic Simulated Annealing and its Application to Neural Network Synaptic Weight Selection

Justin R. Fletcher

SYNAPTIC ANNEALING: ANISOTROPIC SIMULATED ANNEALING AND ITS APPLICATION TO NEURAL NETWORK SYNAPTIC WEIGHT SELECTION

THESIS

Justin Fletcher, First Lieutenant, USAF

AFIT-ENG-MS-16-J-060

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-16-J-060

SYNAPTIC ANNEALING: ANISOTROPIC SIMULATED ANNEALING AND ITS APPLICATION TO NEURAL NETWORK SYNAPTIC WEIGHT SELECTION

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

Justin Fletcher, B.S.C.E.
First Lieutenant, USAF

June 2016

AFIT-ENG-MS-16-J-060

SYNAPTIC ANNEALING: ANISOTROPIC SIMULATED ANNEALING AND ITS APPLICATION TO NEURAL NETWORK SYNAPTIC WEIGHT SELECTION

THESIS

Justin Fletcher, B.S.C.E.
First Lieutenant, USAF

Committee Membership:

Dr. Michael J. Mendenhall
Co-Chairman

Dr. Gilbert L. Peterson
Co-Chairman

Dr. Mathew C. Fickus
Committee Member

AFIT-ENG-MS-16-J-060

Abstract

Machine learning algorithms have become a ubiquitous, indispensable part of modern life. Neural networks are one of the most successful classes of machine learning algorithms, and have been applied to solve problems previously considered to be the exclusive domain of human intellect. Several methods for selecting neural network configurations exist. The most common such method is error back-propagation. Back-propagation often produces neural networks that perform well, but do not achieve an optimal solution.

This research explores the effectiveness of an alternative feed-forward neural network weight selection procedure called synaptic annealing. Synaptic annealing is the application of the simulated annealing algorithm to the problem of selecting synaptic weights in a feed-forward neural network. A novel formalism describing the combination of simulated annealing and neural networks is developed. Additionally, a novel extension of the simulated annealing algorithm, called anisotropicity, is defined and developed.

The cross-validated performance of each synaptic annealing algorithm is evaluated, and compared to back-propagation when trained on several typical machine learning problems. Synaptic annealing is found to be considerably more effective than traditional back-propagation training on classification and function approximation data sets. These significant improvements in feed-forward neural network training performance indicate that synaptic annealing may be a viable alternative to back-propagation in many applications of neural networks.

Acknowledgements

This thesis is the product of hundreds of small investments, made by dozens of people, over the past two years.

To Eric, Bill, Ben, Jason, Olivia, Matt, Jose, James, Geoff, Ian, and Jacob: you are the best friends I could ask for. Your guidance and encouragement made this work possible. During the hardest times, I always knew that I could go to the quad for a laugh and a talk.

I thank Dr. Mendenhall for his tireless support, tolerance of long meetings, and boundless enthusiasm for research. Without your guidance, I would not be on the path I'm on today, and this thesis wouldn't exist.

Finally, and most importantly, I thank my loving wife. You are my best friend and greatest supporter. I wouldn't have made it through this without you. The success of this research is the direct result of many meals served at my desk, hundreds of small encouragements when I was overwhelmed, and a thousand gentle reminders to "get back to work." Thank you, for everything.

Justin Fletcher

Table of Contents

List of Figures

Figure Page

List of Tables

SYNAPTIC ANNEALING: ANISOTROPIC SIMULATED ANNEALING AND ITS APPLICATION TO NEURAL NETWORK SYNAPTIC WEIGHT SELECTION

I. Introduction

Machine learning algorithms are ubiquitous in modern life; rarely is any significant decision made without first consulting a machine learning algorithm. From the most trivial of decisions, such as an individual deciding where to eat for dinner or choosing an entertainment source [1], to the most consequential, such as deciding where the nation should invest its defense budget, machine learning algorithms influence and inform choices. Through vectors such as search engines [2], content recommenders [1], artificial intelligence in adversary behavior models, actuarial models in insurance estimation, market prediction agents, and many more [3], machine learning permeates nearly every modern human endeavor. It seems reasonable to speculate that most people consult at least one machine learning algorithm every day before breakfast, if for no other reason, than to know what the weather will be like. The prevalence of these algorithms attests to their usefulness. Machine learning algorithms have made it possible to analyze data on a scale which would have been impossible in their absence, and thereby enabled the advancement of science in several fields [4]. They have also spawned entire business models which add value for consumers and producers alike.

There are many machine learning algorithms [5, 6]. This thesis focuses on one: neural networks, and in particular, the feed-forward neural network (FFNN). Neural networks are solving problems previously thought to be beyond the capability of any machine; with every advance, the range of activities exclusively in the purview of human intellect recedes further. Today, neural networks are capable of, among other things, extracting patterns from data, classifying observations of data based on past exposure to similar data, and approximating arbitrarily complex functions. The relatively recent advent of deep networks has hastened progress in the field [7]. Deep neural networks have been constructed which are able to play some video games with performance exceeding that of the professional human player [8], and can determine the location an image was taken using only the image [9]. These results are encouraging, as they indicate that neural networks may be capable of obtaining still greater feats of reasoning.

The central concern of the field of neural networks is the problem of constructing a network that is able to do some useful task. There are many variants of neural networks, but many of them share a common learning algorithm: back-propagation. Though effective, all neural networks employing back-propagation share a common limitation, known as the local minima problem. Because back-propagation is a local gradient descent algorithm, it is possible that the algorithm will descend into a region of the cost surface which is a local minimum, rather than the global minimum. If this occurs, and there is no procedure in place to enable an escape from the minima, it will not be possible to improve the performance of the neural network further. Some techniques which minimize the impact of the local minima problem, such as momentum [10], have become standard in most back-propagation implementations. In order to overcome the local minima problem entirely, an alternative neural network weight selection algorithm, called synaptic annealing, is developed in this thesis.

1.1 Problem Statement

The goal of this thesis is to design a set of formalisms describing the operation of simulated annealing on the synaptic weights of a FFNN, to construct a software system which realizes those formalisms, to apply that software system to several machine learning problems, and to evaluate the performance of the system on those problems. While previous work has explored this synthesis of algorithms [11, 12], a review of the literature reveals no work that analyses the combination of simulated annealing and FFNNs to the depth that the topic is explored in this thesis. Specially, this thesis considers many variants of simulated annealing, as well as a novel extension of the simulated annealing algorithm, and compares their resultant performance on several machine learning problems.

1.2 Scope

The number of possible variations of implementation that can be attempted to accomplish the goals set out in the preceding section is very large. As such, it is important to explicitly state the scope of the work addressed in this thesis. Though the scope of the work is relatively broad, the work is limited to the accomplishment of the following objectives:

- Construct a consistent set of formalisms that unambiguously describes the application of simulated annealing to the problem of neural network weight selection.

- Implement all necessary software systems needed to realize a system that performs simulated annealing to select the weights of a FFNN, such that some cost function is minimized.

- Design a series of experiments that explore the cost-function minimization performance of the system on a set of standard machine learning data sets.

- Compare the final performance of each of the constructed algorithms to that of a well-constructed back-propagation algorithm, in order to ensure that an improvement in classification accuracy has been achieved.

1.3 Document Organization

This document is divided into five chapters. Chapter II is a literature review comprising discussions of the historical origins of artificial neural networks as well as some modern applications, simulated annealing and the many variants thereof, and all previous work applying simulated annealing to artificial neural network weight selection. Chapter III presents the methodology used in this thesis to apply simulated annealing to neural network weight selection. The formalisms describing the proposed annealing system are constructed abstractly, then applied to the problem of weight selection. Finally, some preliminary performance exploration is conducted in order to characterize the weight space traversal characteristics of each annealing system. Chapter IV describes the design and results of several experiments which characterize the performance of each synaptic annealing algorithm. Finally, Chapter V concludes the thesis with a brief overview of findings and contributions, as well as suggested future work.

II. Background

This chapter serves as a review of the physical and computational concepts foundational to this thesis. A broad overview of artificial neural networks and the application and history thereof is presented. Next, several formulations of simulated annealing are described, along with a summary of related works and a description of the physical inspiration for the algorithm. The chapter concludes with a very brief overview of the quantum mechanics, with emphasis placed on those concepts used throughout the document.

2.1 Artificial Neural Networks

It has long been recognized that the capacity of biological information processing systems to flexibly and quickly process large quantities of data greatly exceeds that of sequential computing machinery. [10] This information processing capability arises from the complex, nonlinear, parallel nature of biological information processors. The family of models designed to replicate this powerful information processing architecture are collectively called artificial neural networks (ANNs). In the most general sense, ANNs are parallel distributed information processors [10] comprising many simple processing elements. Networks store information about experienced stimuli in the form of connection strengths and network topology, and can make that information available. In such a network, inter-neuron connection strengths are used to encode information, and are modified via a learning strategy. ANNs are characterized by three features: a network topology or architecture, an activation function, and a learning strategy; each is discussed in the following sections. First, however, an abbreviated history of ANNs is provided.

Historical Overview.

The study of ANNs began with a 1943 paper [13] by McCulloch and Pitts. In this paper, McCulloch and Pitts united, for the first time, neurophysiology and formal logic in a model of neural activity. This landmark paper marked the beginning of, not only the computational theory of neural networks, but also the computational theory of mind, and eventually led to the notion of finite atomata [14]. In [13] McCulloch and Pitts introduced a very simple model of a neuron, which acted as a threshold-based propositional logic unit. Significantly, McCulloch and Pitts showed that a network of their neuron models, interconnected, could represent a proposition of arbitrarily-high complexity. Said differently, a network of the neuron models described in [13] can represent any logical proposition. These models, often called McCulloch-Pitts neurons, permit only discrete input values that are summed and compared to a threshold value during a fixed time quantum, and do not posses any learning mechanism. McCulloch-Pitts neurons are able to incorporate inhibitory action, but the action is absolute and inhibits the activation of the neuron without regard to any other considerations. The McCulloch-Pitts neuron model is of theoretical significance, but cannot be applied to practical problems.

Though McCulloch and Pitts made mention of learning in their 1943 paper, thirteen years would pass before the learning concept was formalized into a mathematical and computational model. In 1956 Rochester, et al. [15] presented the first attempt at using a physiologically-inspired learning rule to update the synaptic weights of a neural network. This model was based on the correlation learning rule postulated in 1949 by Hebb[1]. In his book *The Organization of Behavior*, Hebb suggested that synaptic plasticity, the capacity of synaptic strengths to change, is driven by metabolic and

[1] It should be mentioned that, while Hebb was the first to postulate the correlation learning rule as it relates to neurons and synaptic connection strength, the abstract rule was foreshadowed as early as 1890 by William James [16] in Chapter XVI of *Psychology (Briefer Course).*

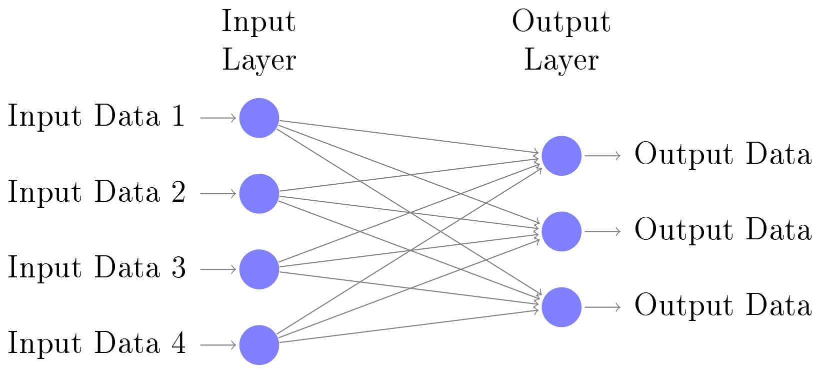


Figure 1. A simple perceptron.

structural changes in the both neurons near the synaptic cleft [17] such that if two cells often fired simultaneously, the efficiency with which they cause one another to fire will increase. This efficiency is now called a synaptic weight. Rochester et. al. showed that the addition of variable synaptic weights alone was not sufficient to produce a network capable of learning; the weights must also be capable of assuming inhibitory values.

The next major contribution to the field would come in 1958 with Rosenblatt's introduction of the simple perceptron [18]. The perceptron, shown in Fig. 1, was the first [19] well-formed, computationally-oriented neural network. Crucially, and unlike most preceding neural models, the model Rosenblatt presented in his 1958 paper was associative. That is, the model learned to associate stimuli with a response. This learning is accomplished by modifying the synaptic weights such that the difference between an input pattern and the desired output pattern is minimized. The responsibility for the error, or difference between the correct and computed output patterns, is divided among the weights in proportion to their magnitude. Thus, large synaptic weights are reduced more than small synaptic weights for a large, positive error. This weight update strategy is represented mathematically as:

$$w_i(t+1) = w_i(t) + \alpha(t)(d_j - y_j)x_j \tag{1}$$

where $w_i(t)$ the synaptic weight for feature i at discrete time t, α is the tunable learning rate parameter, d_j is the desired output, y_j is the computed output, and x_j is the input pattern. This method constitutes a form of reinforcement learning.

Rosenblatt's perceptron was found to be successful at predicting the correct response class for stimuli only if the responses were correlated. It was not until Block's 1962 publication [20] that the reason for this observed performance was elucidated. Block presented two key findings: first, that simple perceptrons require linearly separable classes to achieve perfect classification and second, the perceptron convergence theorem [20]. Linear separability is the ability of the response classifications to be separated by a hyperplane in the n-dimensional space of the input stimuli to which they correspond. The requirement of linear separability arises directly from the way in which the output of a perceptron response unit is calculated. The output of a simple perceptron response unit is given by the hard limiter function:

$$y_j = \begin{cases} -1 & \sum_{i=1}^{n} w_{i,j} x_i \leq \Theta \\ +1 & \sum_{i=1}^{n} w_{i,j} x_i > \Theta \end{cases} \tag{2}$$

where y_j is the response value of response unit j, $w_{i,j}$ is the synaptic weight of the connection between activation unit i and response unit j, x_i is the activation value of activation unit i, and Θ is the threshold value of the perceptron. Block's crucial observation was that the form of the summation in the response determining equation is isomorphic to a hyperplane in an n-dimensional space. Thus, in order for the perceptron to achieve perfect classification, a hyperplane must be able to separate them in the n-dimensional input space. The corollary of this observation is the perceptron convergence theorem. The theorem proves that, for some learning rules, if a perfect classification is possible it will be found by the perceptron. Specifically, the class of learning rules which were found effective were those that do not change synaptic weights when a correct classification occurs. While the condition does ensure convergence, it often causes very slow convergence, as the synaptic weights change much more slowly when only a small number of samples remain misclassified. Considerably faster guaranteed convergence can be achieved using an error gradient descent learning rule [21], as described by Widrow and Hoff.

In 1969 Minksy and Papert published *Perceptrons* [22], a book on mathematics and theory of computation. In this book, Minsky and Papert mathematically and geometrically analyzed the limitations inherent in the perceptron model of computation. The authors reasoned that the each response unit of Rosenblatt's perceptrons was actually computing logical predicates about the inputs it received, based on the observation that response units can either be active or inactive. This analytical framework allowed the authors to construct unprecedented geometric and logical arguments about the computational capabilities of a perceptron. They found that there were several classes of problems which were unsolvable by linear perceptrons [22]. In the final chapter of *Perceptrons*, Minsky and Papert extended their judgments regarding the ineffectiveness of single-layered perceptrons to all variants of perceptrons, including the multi-layered variety. This conjecture would turn out to be one of the most significant of the entire book, as it likely resulted in a reduction of funding for neural network research [19] which lasted for several years. However, this judgment was incorrect.

While it is true that the pace of development in the field of neural networks slowed considerably after the publication of *Perceptrons*, there was still progress made during the 1970s. In 1972, both Anderson [23] and Kohonen [24] published models of what would come to be known as linear associative neural networks, which are a generalization of Rosenblatt's perceptron. As with the perceptron, neurons in a linear associative neural network compute their output by summing the product of each input signal and the synaptic weight associated with that input. Unlike the perceptron, the output of these networks is proportional to this sum, rather than a binary value computed by applying a threshold function to the sum. Though still unable to achieve perfect classification on many classes of problems, these networks were able to successfully associate input patterns with output patterns.

The decade also saw the advent of self-organized maps, which are a type of competitive learning neural network. Self-organization in neural networks was first demonstrated by van der Malsburg [25] published in 1973. This paper analyzed the response of simulated cortical cells to a simulated visual stimulus. The paper is interesting both for the complexity of the neural model developed, and because it contained the first direct comparison between computer simulation and physiological data [19].

Throughout the decade progress was also made in the understanding of the physiology of biological neural networks. Of particular interest are those papers describing the lateral retinal system of *Limulus polyphemus*, the horseshoe crab. *Limulus* features prominently in the neurophysiological research because of the ease with which experiments may be conducted on its compound lateral eye. Several works were published on the *Limulus*, perhaps the most significant of which came near the end of the decade with the 1978 publication of a paper describing the dynamics of the retina of a *Limulus* when exposed to moving stimuli [26]. In this paper, the *Limulus* eye was analyzed as a linear system, and the results of this analysis were compared to the actual response of the system to the input pattern. The agreement between the linear[2] model and the biological output signals was found to be in excellent agreement [26]. This finding was interesting for the purposes of perceptron simulation, but was ultimately found not to hold for larger collections of neurons.

[2] Linear, in this context of this system, means that the output of the system when presented with the sum of a set of inputs is equivalent to the sum of the outputs of the system when presented with each input individually.

Several events conspired to create a reinvigoration of neural network research in the early 1980s. In 1982, John Hopfield published *Neural networks and physical systems with emergent collective computational abilities* [27]. This momentous work is regarded by many to be the beginning of the renaissance of neural network research [19], and contains many novel insights. Hopfield begins the paper differently than past researchers. Rather than proposing a learning rule or network topology and then evaluating the results of this proposition, Hopfield begins by considering an alternative purpose for a neural network. Hopfield suggests that the network be thought of as a means to develop locally stable points, or attractors, in a state space. The state space comprises the set of states which are the activation value of each neuron. Thus, learning should be the process of modifying the synaptic weights such that they cause the system to flow into local attractors which represent the desired output. In such a model, a noisy or incomplete input would result in an activation pattern that resides on a gradient in the state space. The neural network would then change the activation pattern in such a way as to move the system down the gradient into the attractor state. Hopfield suggests that this process is a general physical description of the concept of content-addressable memory.

Hopfield then proposes a network architecture to achieve this behavior [27]. The chosen model is one which has binary neural output values and recurrent connections. Neural networks of this types are now called Hopfield networks. Like Rosenblatt's original perceptron model, the neurons used in Hopfield's work had a non-linear activation function. The network topology was recurrent, with the restriction that no neuron could provide input to itself. Hopfield adopted a variation of Hebb's learning rule to update the synaptic weights.

In Hopfield's network model [27], the connection strength between two neurons i and j is denoted as T_{ij}, and the activation status of a neuron i is denoted as

V_i. T is therefore the connection matrix of the neural network, with each element representing an individual connection strength and zeros along the diagonal. It is from this organization of the connection strengths that one of the most important insights of this thesis originates. Hopfield recognized that, in the special case of the model in which $T_{ij} = T_{ji}$, a quantity E could be defined such that

$$E = -\frac{1}{2}\sum_{i \neq j}\sum T_{ij} V_i V_j, \tag{3}$$

where E is a quantity analogous to the energy of an Ising model of a spin glass. The change in E as a result of a change in one of the activation values, V_i (in the following equation), is then represented as

$$\Delta E = -\Delta V_i \sum_{j \neq i} T_{ij} V_j. \tag{4}$$

From Eq. (4), it is clear that any change in V_i will reduce the value of E. This decrease in E must necessarily continue until some local minimum of the value of E is reached[3]. Here, Hopfield observed that this case is isomorphic with an Ising model, referencing the statistical mechanical model of magnetic spins. In this isomorphism, the quantity E maps to the energy of a physical system described by an Ising model. It is difficult to overstate the importance of this observation. It both provided a novel mechanism by which physical theory could be applied to neural networks, and legitimized the study of neural networks as a physical system, encouraging many physicist to join in the development of the theory.

Hopfield constructed a model of the system described in the paper, and presented it with random input patterns[4]. He found that the network can indeed recall a

[3] An identical conclusion would be reached if V_j was changed instead of V_i. It is merely a matter of convention.

[4] Hopfield calls these input patterns entities or *Gestalts*.

small number of patterns, on the order of approximately 15 percent of the network dimensionality, before the recall error becomes significant.

In 1985, the work by Ackley et al. [28] extended the neural network model proposed by Hopfield[5]. Hopfield networks are deterministic with respect to energy; by definition any change in a Hopfield network always reduces the energy of the system or leaves it the same. This is a useful property if it is acceptable to find one of many local minima, or attractors. However, if a single, global minima in the state space is sought, this model is likely to converge prematurely to a local attractor state. In order to surmount this limitation, Ackley et. al. modified the Hopfield neural model to activate *stochastically.* The probability of state transition, p, is given by

$$p = \frac{1}{1 + e^{-\Delta E/T}} \tag{5}$$

where ΔE is the change in energy of the system resulting from a transition to a new state and T is the artificial temperature of the system. Thus the relative probability, P_α/P_β of moving to either of two arbitrary global states, α and β, is defined as

$$\frac{P_\alpha}{P_\beta} = e^{-(E_\alpha - E_\beta)/T} \tag{6}$$

which is a form isomorphic to the Boltzmann distribution. Thus, a neural network with transition probabilities described by Eq. (5) is called a Boltzmann machine. The effect of probabilistic state transitions of this form is that state transitions from low energy states to higher energy states are possible, thereby allowing the system to *escape* local minima.

Inspection of Eq. (5) and Eq. (6) reveals that the probability of transition is determined by both ΔE and T. A large value of ΔE, which corresponds to a large increase

[5] Though a Hopfield network was used for the work done by Ackley, Hinton, and Sejnowski it is not necessary to use a network with recurrent connections.

in total energy, will decrease the probability of transition. Conversely, a large value of T will increase the probability of transition for any arbitrary value of ΔE. The systems artificial temperature therefore acts as a tuning mechanism for the exploration of the state space. A high artificial temperature results in greater exploration of the state space, but will result in less local gradient descent and therefore may cause the system to depart the attractor basin of a minima, which may be the global minimum. A low artificial temperature may cause premature convergence. Ackley et. al. [28] solved this tuning problem by recognizing a deep connection to another concept born of statistical mechanics: simulated annealing. Simulated annealing decreases the artificial temperature of a system slowly over the course of a simulation, and as a result, increases the likelihood that the final state of the system will be the ground state, which is to say the global minimum. This algorithm is discussed in detail in Sec. 2.2.

With a procedure for finding the global minimum of a cost surface embedded in a state space in place, the authors in [28] proceeded to construct state spaces for which the global minimum was of interest. One way to construct such a state space is to include hidden units in the neural network. Hidden units are neural units which are neither input nor output units. These hidden units allow the network to solve interesting problems that are out of reach of simple associative neural networks[19]. However, like all neurons in any neural network, the connection weights of these neurons must be modified in order for the network to learn. It is not immediately clear how the synaptic weights of hidden unit can be modified to account for the performance of the network. This deficiency is often called the credit assignment problem [29]. The application of simulated annealing in Boltzmann machines avoids the problem of assigning credit to hidden units, thereby enabling their inclusion in the model. This is historically significant because it was the first successfully-implemented multi-layered neural network [10].

The next major development in neural networks came in 1986 with the introduction of the back-propagation algorithm. Though the formalisms required involved in this algorithm had been developed earlier [30, 31], and the method was simultaneously discovered independently by two other groups [32, 33], it was Rumelhart et al. that applied the algorithm to machine learning [34]. Back-propagation can be thought of as a generalization of the gradient descent[6] algorithm presented by Widrow and Hoff [21], which includes the errors associated with connection strength of hidden units, or internal representation units. A detailed discussion and derivation of the back-propagation algorithm is included in Section 2.1. Though back-propagation in multilayered perceptrons cannot be guaranteed to find an exactly correct solution, the algorithm is demonstrably capable of solving difficult and interesting problems, thus disproving the speculation of Minsky and Papert in [22].

With the advent of Boltzmann machines and back-propagation, it became possible to analyze the properties and capabilities of multilayered neural networks. In 1989 Cybenko showed that a multilayer feed-forward neural network (FFNN) with nonlinear activation function is, in principle, capable of approximating any continuous function [35]. This finding is striking because it implies that, given the correct learning rule and a sufficiently large network, a multilayer neural network can learn a pattern of arbitrary complexity.

Network Topology.

The topology of a neural network describes the way in which the individual processing units are interconnected. There are only a few broad classes of topology, each of which has different properties. A FFNN consists of several successive layers of processing elements. The input pattern is provided to the first layer of the network.

[6] The gradient descended in this context is the gradient of the error surface in the space of synaptic weights, not the gradient of the activation state surface, as with a Hopfield network.

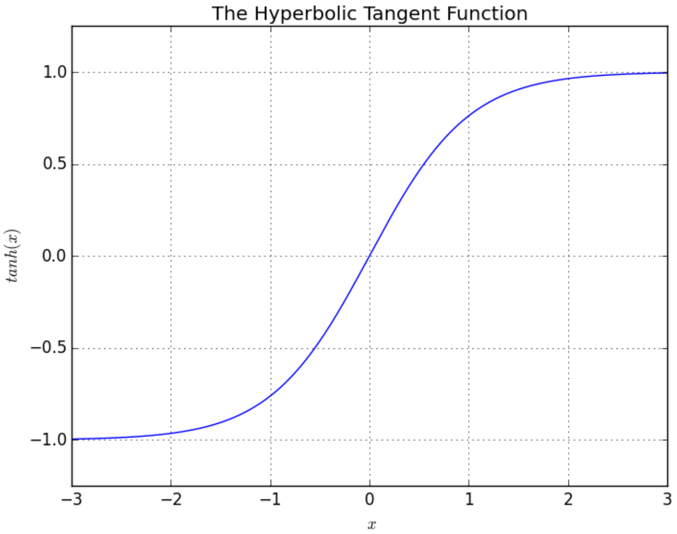
This layer, often called the input layer, contains one processing element for each input pattern element. Each input pattern element is provided to exactly one input layer processing element, and each input layer processing element is connected to every processing element in the next layer. The next layer in the network is called the first hidden layer. The input layer is fully connected to the first hidden layer. This means that each processing element in the input layer is connected to each element in the first hidden layer. This fully-connected organization is then repeated between each successive hidden layer. The final layer in the FFNN topology is the output layer. There is one processing element in the output layer for each element in the output pattern. The final hidden layer is fully connected to the output layer. Feed-forward neural networks are therefore directed, acyclic L-partite graphs, where L is the number of layers in the network. The graph describing the topology of a neural network is often called the *underlying graph* of the network. Generally, these graphs are often organized as a left-right model with inputs on the left and outputs on the right.

Activation Functions.

An activation, or transfer function, is the transform applied to the sum of weighted inputs in each processing element. There are myriad activation functions used in the neural network literature. In this thesis, the hyperbolic tangent function is used, and is shown in Fig. **??**.

Learning Strategies.

Very simple neural networks may be constructed by hand to solve simple problems. To encode more complex mappings, automated procedures are used to modify neural network parameters. These procedures, called learning strategies, generally involve the modification of synaptic weights in the neural network, such that the modifica-



tions minimize a cost function. This section discusses two such learning strategies. This first algorithm, back-propagation, is a widely-used gradient descent method for minimizing the error in the output of a neural network. The second, which is the topic of this thesis, is the application of metaheuristic algorithm, simulated annealing, to the problem of neural network synaptic weight selection.

Back-Propagation Training.

The most widely used learning strategy for FFNNs is error back-propagation [10]. In back-propagation, the synaptic weights of the network are adjusted proportionally to their contribution to the error in the output of the network, with respect to the desired output. The process of determining how much an individual synaptic weight contributed to the output error of the network is often called the *credit assignment problem*; the back-propagation algorithm is one solution to the credit assignment problem for FFNNs.

In the literature discussing the derivation of the back-propagation algorithm, a standard mathematical notation describing the operation of FFNNs is used. In this notation, the input to the activation function of a neuron j is given by

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n), \tag{7}$$

where w_{ji} is the synaptic weight of the connection from neuron i to neuron j, $y_i(n)$ is the output signal from neuron i, and m is the total number of inputs to neuron j. Therefore, the output of neuron j given the nth input pattern, $y_j(n)$, is given by

$$y_i(n) = \varphi_j(v_j(n)) = \varphi_j\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right). \tag{8}$$

The back-propagation algorithm begins begins by defining the error of an individual output neuron, j, which is given by

$$e_j(n) = d_j(n) - y_j(n), \tag{9}$$

where $d_j(n)$ is the desired output value for output neuron j and $y_j(n)$ is the actual output from output neuron j, given the nth input pattern. This error quantity, which is defined for each output layer neuron, is then combined to form and instantaneous error energy term for the entire network, given by

$$\mathcal{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \tag{10}$$

where C is the set of integer labels indicating the output neurons, and $e_j(n)$ is the error of output neuron j given the nth input pattern. The error energy is therefore simply the sum os squared errors, over all output neurons.

The objective of the back-propagation algorithm is to minimize $\mathcal{E}$ by modifying each synaptic weight in the network according to its share of responsibility for $\mathcal{E}$. Thus, given an error value, $\mathcal{E}$, the back-propagation algorithm constructs a correction factor, $\Delta w_{ji}(n)$, which is the quantity by which the synaptic weight w_{ji} must be changed in order to minimize $\mathcal{E}(n)$ for the nth input pattern. Here, $\Delta w_{ji}(n)$ should change the weight $w_{ji}(n)$ in such a way as to reduce the error, which is equivalent to descending the gradient of the error in the weight space. Observe that the partial derivative is given by

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}. \tag{11}$$

The partial derivative given in Eq. (11) gives the sensitivity of $\mathcal{E}(n)$ with respect to an individual synaptic weight $w_{ji}(n)$. Thus, calculating the instantaneous value of Eq. (11) yields the change in the weight $w_{ji}(n)$ which most reduces $\mathcal{E}(n)$ for the nth input pattern. A learning rate parameter, α, is introduced to control the rate of descent. Combining these values yields the desired correction factor, $\Delta w_{ji}(n)$, often called the *delta rule*, which is given by

$$\Delta w_{ji} = -\alpha \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}. \tag{12}$$

The negation of the partial derivative in Eq. (12) is indicative of *gradient descent*.

In order to construct a functional form of Eq. (12), suitable of algorithmic implementation, further development is required. Applying the chain rule to Eq. (11), the expression

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{13}$$

is constructed. A functional form of each partial derivative in Eq. (11) may be constructed from the mathematical framework used to describe FFNNs. Differentiating

Eq. (10) with respect to $e_j(n)$ yields

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n). \tag{14}$$

Differentiating Eq. (9) with respect to $y_j(n)$ yields

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1. \tag{15}$$

Differentiating Eq. (8) with respect to $v_j(n)$ yields

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)). \tag{16}$$

Differentiating Eq. (7) with respect to $w_ij(n)$ yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \tag{17}$$

Substituting Eq. (14), Eq. (15), Eq. (16), and Eq. (17) into Eq. (13) yields

$$\Delta w_{ji} = \alpha e_j(n)\varphi_j'(v_j(n))y_i(n), \tag{18}$$

which is often expressed as

$$\Delta w_{ji} = \alpha \delta_j(n) y_i(n), \tag{19}$$

where $\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$; $\delta(n)$ is called the *local gradient.* Eq. (19) is the desired functional form of the delta rule. Observe, however, that $\delta(n)$ depends on $e_j(n)$ which in turn depends on $d_j(n)$, which is only defined for output neurons. Thus, the back-propagation algorithm must compute the local gradient differently, depending on where the neuron for which the incident synaptic connections are being updated resides in the network. For output layer neurons, the delta rule is stated in Eq. (19). For hidden layer neurons the computation is somewhat more complex.

There is no explicit error signal for any neuron in the hidden layers, thus, one must be constructed from those neurons that do have an error signal. Eq. (13) dictates that the local gradient for a hidden layer neuron j is given by

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \tag{20}$$

which, in turn, yields

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)} \varphi_j'(v_j(n)) y_i(n). \tag{21}$$

In order to calculate the partial derivative $\partial E(n)/\partial y_j(n)$, the error from an output neuron, given in Eq. (10), must be used. Differentiating Eq. (10), where j in that equation is relabeled to k to avoid confusion, yields

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}. \tag{22}$$

Additionally, observe that Eq. (9) becomes

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k'(v_k(n)). \tag{23}$$

which yields

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)) \tag{24}$$

by differentiation with respect to $v_k(n)$. Further, observe that by relabeling Eq. (7)

in terms of k, an expression which gives the input to output neuron k from hidden neuron j is obtained:

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_i(n). \tag{25}$$

Differentiating Eq. (25) with respect to $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_i(n)} = w_{kj}(n). \tag{26}$$

Substituting Eq. (26) and Eq. (24) into Eq. (22) the desired partial derivative, $\partial E(n)/\partial y_j(n)$, is recovered and is found to be

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi_k'(v_k(n)) w_{kj}(n), \tag{27}$$

which reduces to

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) w_{kj}(n). \tag{28}$$

Finally, substituting Eq. (28) into Eq. (21) yields

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n), \tag{29}$$

which gives the local gradient of the error for hidden layer neuron j.

Thus, a function specifying the weight correction factors for synaptic weight incident to neurons in the hidden and output layers is constructed. This delta rule function is given by

$$\Delta w_{ji} = \alpha \delta_j(n) y_i(n), \tag{30}$$

where

$$\delta_j(n) = \begin{cases} e_j(n)\varphi_j'(v_j(n)) & j \in C \\ \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & j \notin C \end{cases}, \tag{31}$$

where C is the set of output neurons.

The back-propagation algorithm updates the weights of the FFNN by propagating an input pattern though the network, then propagating the resultant error signal backward through the network, applying Eq. (30) and Eq. (31) to each synaptic weight. The back-propagation algorithm is effective, but also has limitations. The algorithm descends the gradient of the error surface in order to minimize the error. If the error surface contains local minima, the back-propagation algorithm can become stuck in these minima, because there is no local gradient to follow, thereby prematurely resulting in a neural network which produces a greater than minimal error.

Simulated Annealing Training.

The first application of simulated annealing (SA) to the training of neural networks was accomplished by Engle in [12], with limited success. Engle's work involved discrete-weight neural networks, and was limited in applicability. Since, several publications [11, 36, 37] have described the application of SA to neural network weight selection. In [11], for example, multivariate simulated annealing is used to reduce the generalization error of a neural network trained using other gradient descent methods.

2.2 Related Works in Simulated Annealing

Simulated annealing is a stochastic optimization algorithm that can be used to find the global minimum of a cost function mapped from the configurations of a combinatorial optimization problem. The concept of simulated annealing was introduced by Kirkpatrick et al. [38] as an application of the methods of statistical mechanics to the problem of discrete combinatorial optimization. Specifically, simulated annealing is an extension of the Metropolis-Hastings [39] algorithm, which can be used to estimate the ground energy state of a many-body systems at thermal equilibrium. Kirkpatrick, et al. applied the Metropolis-Hastings algorithm sequentially, with decreasing temperature values in order to approximate a solid slowly cooling to low temperatures. Later work by Goffe [40], Corana, et al. [41], and Lecchini-Visintini et al. [42] extended SA to the continuous domain.

In the most general terms, SA is a local search algorithm through the problems solution space, $\boldsymbol{\mathcal{S}}$, which is the set of all possible solutions, $\boldsymbol{s}$, of the problem, where $s \in \boldsymbol{\mathcal{S}}$. The search is conducted by generating new solutions to the problem by applying a neighborhood function, $\boldsymbol{\mathcal{N}}$, to the current solution; the neighborhood function specifies the way in which a solution is transformed to yield a new solution, or neighbor solution, and is generally problem dependent. To apply SA to an optimization problem it is necessary that each possible solution of the problem be characterized by a cost function, $\mathcal{C}$, where $\mathcal{C}$ is a mapping $\mathcal{C} : \boldsymbol{\mathcal{S}} \to \mathbb{R}$. Because $\mathcal{C}$ is a function on $\boldsymbol{\mathcal{S}}$, it is said that the cost function forms a cost surface in the solution space. During each iteration of the algorithm, a new solution $\boldsymbol{s}'$ is generated using $\boldsymbol{\mathcal{N}(s)}$, and the cost of that solution, $\mathcal{C}(\boldsymbol{s}')$, is determined. The change in cost associated with moving from the current solution to the neighbor solution is given by

$$\Delta C = \mathcal{C}(\boldsymbol{s}') - \mathcal{C}(\boldsymbol{s}).$$

The quantity ΔC is then used in conjunction with an artificial temperature parameter to determine if the newly-generated neighbor solution is to become the current solution. The artificial temperature parameter is specified by the temperature schedule, $T(t)$, where t is the number of simulation iterations completed. Though t often denotes a continuous value, in this case t is a discrete, nonnegative integer value. The temperature controls the probability of the system moving to a higher cost solution, thereby enabling the algorithm to escape local minima on the cost surface. In the parlance of SA [38], a system at its maximum temperature is said to be *melted.* In the melted state, most neighbor solutions are accepted by the algorithm. Analogously, a system that has a temperature of zero, which indicates that the algorithm cannot move to any higher-error state, is said to be *frozen.* Note that a frozen system may still be perturbed into a lower-energy state. The notions of freezing and melting enter the SA algorithm in the form of the acceptance criterion, which determines if a newly-generated solution is to become the current solution. The most commonly used acceptance criterion is the Metropolis criterion [39]. The probability of transitioning to a state, P, given the ΔC associated with that transition is given by:

$$P = \begin{cases} 1, & \Delta C \leq 0, \\ e^{-\frac{\Delta C}{T(t)}}, & \Delta C > 0. \end{cases} \tag{32}$$

The probability of moving to a solution that is higher in cost than the current solution is derived from the statistical mechanical probability of traversing a potential energy barrier by thermal fluctuations. When considering only the influence of classical thermal fluctuations in particle energy levels, the probability of a particle traversing a barrier of height ΔVat a temperature T is on the order of

$$\mathcal{P}_t = e^{-\frac{\Delta V}{T}}. \tag{33}$$

The SA algorithm is presented in Algorithm 1.

Algorithm 1 This algorithm describes a metaheuristic method for identifying the global minimum of a cost surface defined by $\mathcal{C}$. *Inputs:* $\boldsymbol{s}_0$, an initial solution from which subsequent solutions will be produced. $\mathcal{C}$, a cost function defined on the solution space. T, a temperature schedule function. ϵ, a minimum temperature value which, once attained, causes the algorithm to halt; values of ϵ less than or equal to 0.0 may prevent the algorithm from halting, and are therefore invalid inputs. $\boldsymbol{\mathcal{N}}$, a neighborhood function which produces a neighboring solution from a provided solution. *Outputs:* $\boldsymbol{s}_{opt}$, a solution which corresponds to the minimum cost function value, given the input parameters.

```
procedure SimulatedAnnealing($\boldsymbol{s}_0$, $\mathcal{C}$, $\epsilon$, $\boldsymbol{\mathcal{N}}$, $T$)
    $\boldsymbol{s} \leftarrow \boldsymbol{s}_0$                                        ▷ Initialize a solution.
    $t \leftarrow 1$                                           ▷ Initialize the epochs counter.
    while $T(t) > \epsilon$ do          ▷ Repeat until the temperature is sufficiently small.
        $\boldsymbol{s}' \leftarrow \boldsymbol{\mathcal{N}}(\boldsymbol{s})$                               ▷ Generate a neighbor solution.
        $\Delta C \leftarrow (\mathcal{C}(\boldsymbol{s}') - \mathcal{C}(\boldsymbol{s}))$                          ▷ Compute the cost change.
        if $(\Delta C \leq 0) \vee (\exp(\Delta C/T(t)) > U(0,1))$ then        ▷ Apply Metropolis criterion.
            $\boldsymbol{s} \leftarrow \boldsymbol{s}'$                                    ▷ Accept the new state.
        end if
        $t \leftarrow t + 1$                                   ▷ Increment the epoch count.
    end while
    $\boldsymbol{s}_{opt} \leftarrow \boldsymbol{s}$                                 ▷ Declare the optimal solution.
    return $(\boldsymbol{s}_{opt})$                             ▷ Return the optimal solution.
end procedure
```

Variants of Simulated Annealing.

Since the publication of the original description of SA, several variations of the algorithm, many of which improve its efficiency considerably, have been described. In [43], Szu and Hartley introduced the method of fast simulated annealing (FSA), which incorporates occasional, long jumps through the configuration space, These jumps are accomplished by using heavy-tailed distributions, such as the Cauchy distribution, for the visiting distribution used in the neighborhood function. This provision increases the likelihood of escaping local minima, and reduces the total computational effort required to reach a global minimum. This modification yields a significant decrease in the amount of computation effort required to guarantee that a global minimum is found. Specifically, FSA increases the acceptable temperature decay rate relative to the original specification of SA, classical simulated annealing (CSA). The CSA temperature schedule is given by

$$T_{\mathrm{CSA}}(t) = T_{\mathrm{CSA}}(1)\frac{1}{\ln(t)}, \tag{34}$$

while the FSA temperature schedule is given by

$$T_{\mathrm{FSA}}(t) = T_{\mathrm{FSA}}(1)\frac{1}{t}. \tag{35}$$

This considerable increase in the rate at which the temperature is able to decrease, while still finding a global minimum, significantly reduces the computational effort required to find a global minimum. A detailed proof of the sufficiency of this temperature schedule can be found in [44, 43], and an intuitive explanation based on the configuration space traversal properties of the algorithm can be found in Section 3.1.

Later work by Tsallis and Stariolo [45] generalized both CSA and FSA into a single framework: generalized simulated annealing (GSA). GSA is fundamentally a modification and parameterization of the visiting distribution used to produce new states. This distribution further increases the decay of temperature over training time, yielding a new temperature schedule given by

$$T_{\text{GSA}}(t) = T_{\text{GSA}}(1)\frac{2^{q_T-1}-1}{(1+t)^{q_T-1}-1}, \tag{36}$$

where q_T is a tunable parameter which influences the rate both the rate of temperature decay and the shape of the visiting distribution. The temperature schedule defined in Eq. (36) is shown to be sufficient to enable the location of a global minimum in [46]. GSA is widely-considered to be the state of the art in SA, and has been used with considerable success to solve difficult problems in many fields [47, 48, 49]. Despite this widespread use, a review of the available literature does not reveal application of GSA to the problem of neural network synaptic weight selection; GSA is used to that end in this thesis.

Reannealing.

In [44], Ingber introduced the concept of reannealing; a mechanism that allows for the artificial temperature parameter to occasionally increase. The increase, or rescaling, of the temperature parameter enables the SA algorithm to move to higher cost solutions, effectively restarting the annealing process, but from a configuration space location that is already known to be a local minima. This mechanism allows the SA algorithm to escape from local minima, and thus decreases the simulation time required to achieve convergence to a global minima.

2.3 Relevant Concepts in Quantum Mechanics

Quantum mechanics is the branch of physics concerned with the physical laws of nature at very small scales. Many aspects of physical reality are observable only at these scales. Several variants of simulated annealing described in this document are either inspired by, or are simple models of, quantum mechanical processes. These concepts are very briefly reviewed in this section.

One of the quantum phenomena for which there is no classical analog is potential barrier penetration, also known as quantum tunneling. This phenomenon arises from the probabilistic and wavelike behavior of particles in quantum physics. Tunneling plays a significant role in the behavior of bound and scattering quantum mechanical systems.

A particle with energy E incident upon a potential energy barrier of height $\Delta V > E$ has a non-zero probability of being found in, or past, the barrier. Classically, this behavior is forbidden. The probability of tunneling, $\mathcal{P}_t$, through a step barrier of height ΔV is described by:

$$\mathcal{P}_t = e^{-\frac{w\sqrt{\Delta V}}{\Gamma}} \tag{37}$$

where Γ is the tunneling field strength [50]. Fig. 2 depicts a one-dimensional example of the quantum tunneling of the probability distribution function of the location of a particle indecent upon a potential energy barrier.

2.4 Summary

In this chapter, a review of the history of neural networks is presented, as well as an overview of the essential features of neural networks. The most common training algorithm for FFNNs is derived and one limitation of this algorithm is identified. Relevant literature from other fields is summarized and related to the problem of neural network weight selection. In this thesis, the concepts presented in this chapter will be built upon to construct a neural network training algorithm which is able to overcome some of the limitations of back-propagation.

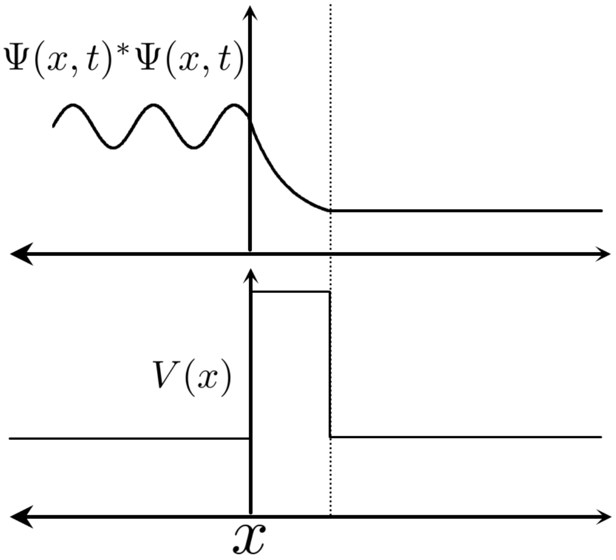


Figure 2. (Top) The probability density function of a generic quantum system in the presence of a potential energy step barrier. There is an exponential decrease in probability through the barrier, and a uniform probability beyond the barrier. (Bottom) A simple step potential in one dimension.

III. Methodology

The application of simulated annealing (SA) to feed-forward neural network (FFNN) weight selection requires the specification of several formalisms and representations linking the two concepts. This chapter presents the representations, of both simulated annealing and FFNNs, adopted in this thesis. These representations are combined into a set of formalisms which describe the application of SA to FFNN weight selection. Initial exploration of the efficacy of these formalisms is conducted.

3.1 Simulated Annealing

Several variations of the SA algorithm are developed and implemented. Each is applied to the problem of selecting synaptic weights in a FFNN in order to maximize the performance of the network; the performance of the network is characterized by its ability to correctly map an input vector to a desired output vector. This section contains an abstract discussion of the SA formulations which are applied to the the weight selection problem; later sections will expound the implementation details specific to the FFNN application of the these SA formulations. For all SA formulations examined in this thesis, the Metropolis acceptance criterion is used. The temperature schedule is determined by the convergence properties of the constructed algorithm. Examining Alg. 1 reveals that the preceding specifications leave only one component unspecified: the neighborhood function, which determines how new trial solutions are generated from the current solution. In the following sections, the neighborhood function is decomposed into two decoupled components, the visiting distribution and anisotropicity policy, and several possible realizations of each are discussed.

The SA algorithm requires the specification of a neighborhood function, $\mathcal{N}$, to produce new solutions from a given solution. The neighborhood function performs the exploration of the solution space, as it specifies new solutions which are either accepted or rejected according to the acceptance criteria. Thus, $\mathcal{N}$ determines how the algorithm traverses the cost surface of the problem. A traversal action, or move, on a surface, may be decomposed into two components: the distance of the move and the direction of the move. These components may be specified independently of one another. The distance of the move on the cost surface has been examined in previous work [44, 43, 45], and is often specified using a *visiting distribution*, which is defined as a probability distribution of transition to a solution over the solution space of the problem. The visiting distribution specifies the magnitude *and* the direction of the move.

In previous work [45], the move distance is applied isotropically in all possible dimensions of travel. In physical science, isotropicity is the phenomenological property of being uniformly applicable in all dimensions. In the context of neighborhood functions, this means that the probability distribution over the solution space is symmetric; that is, that the probability distribution in each dimension is the same. In the following subsections, a method is developed for specifying an anisotropic visiting distribution.

Visiting Distributions.

The neighborhood function visiting distribution is probability a density function over all possible solution states, which specifies the probability of transitioning from the current state of the system to another state. Once a visiting distribution is specified, samples can be drawn from the distribution using inverse transform sampling. These samples may be either drawn from an n-dimensional distribution, where n is the number of free parameters of the problem, or n samples can be drawn and applied to each of the free parameters independently. In this thesis, all visiting distributions are implemented using the latter method. In the following sections, several visiting distributions are presented.

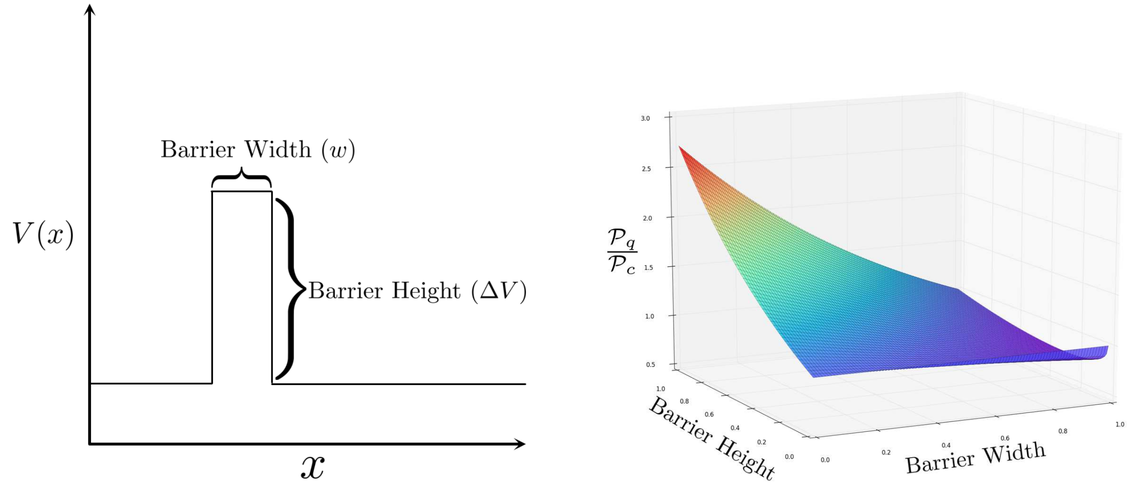
Gaussian Visiting Distributions.

A Gaussian visiting distribution is commonly used in SA. Because it is a light-tailed distribution, and therefore indicates very low probability for all values far from the mean, a Gaussian visiting distribution results in a search that is highly local about the mean, or current solution. An SA algorithm using a Gaussian visiting distribution is often called classical simulated annealing (CSA) [45], both because it is the formulation of SA that was described first, and because the dynamics of the algorithm are isomorphic with those of classical thermodynamics. In this work, a standard normal distribution is used, and is given by

$$g_G(x) = \frac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}}. \tag{38}$$

Cauchy Visiting Distributions.

Local search must occur in order to enable gradient descent, but it introduces a limitation. If the artificial temperature that controls the probability of moving uphill on the cost surface is lowered too quickly, the algorithm can get caught in a local rather than global minima. This is also known as the freezing problem. One way to alleviate this limitation is to construct a neighborhood function that enables the system to escape local minima by means other than hill-climbing. In quantum mechanics, a system that is trapped in a local minimum on a potential energy surface may escape that minima by tunneling through the potential energy surface to a lower energy state. It is possible to construct several visiting distributions which act analogously to quantum tunneling. This can be done by using a visiting distribution that has a non-negligible probability of generating large traversal distances. If the traversal distances generated are sufficiently large, it is possible that the arrived-at solution will be across the cost surface barrier surrounding the local minimum, thus allowing the algorithm to escape the minimum. The term *quantum-inspired visiting distribution* is used in this thesis to describe any distribution possessing this property.



(Left) **(Right)**

Figure 3. (Left) A potential energy barrier on a one-dimensional potential energy surface. (Right) The tunneling probability relative to the probability of traversal due to thermal fluctuation for a step barrier plotted as a function of the height and width of the barrier.

The significant performance gains exhibited by the fast simulated annealing (FSA) algorithm are the result of using a quantum-inspired visiting distribution. In [43], which introduces the FSA algorithm, a Cauchy distribution is used to generate new solutions. Unlike the Gaussian distribution, the Cauchy distribution is heavy-tailed, meaning that it will occasionally produce values which are relatively far from the mean. This property of the distribution has the useful consequence of increasing the probability of escaping a local minima by allowing the algorithm to tunnel through the cost-surface barriers that surround it. This in turn allows for faster convergence relative to CSA. To understand the origin of this advantage, it is instructive to contrast Eq. (33) and Eq. (37). Both describe the same value, but the importance of the width and height of the traversed barrier in the two equations is considerably different. For systems in which quantum tunneling is possible, the probability of penetrating a barrier of height ΔV is increased by a factor of approximately $e^{\Delta V}$, for large values of ΔV. This relationship is depicted graphically in Fig. 3 which shows the probability of barrier traversal for a system which allows quantum fluctuations, divided by the same probability for a system which only considers thermal fluctuations. Fig. 3 (Right) illustrates the fact that physical models which considers quantum effects are much more likely to predict penetration of tall, thin energy barriers than those which only include classical thermal effects.

The general probability density function for the Cauchy distribution is given by

$$g_C\left(x, \Theta_C = \{x_0, \gamma\}\right) = \frac{1}{\pi\gamma} \frac{\gamma^2}{(x - x_0)^2 + \gamma^2}$$

where x_0 is the mean and γ is the shape parameter. In this work, $x_0 = 0$ and $\gamma = 1$, yielding the specific Cauchy distribution given by

$$g_C(x|\Theta = \{0, 1\}) = \frac{1}{\pi x^2 + \pi}. \tag{39}$$

Generalized Simulated Annealing Visiting Distribution.

The most sophisticated form of SA is generalized simulated annealing (GSA), described by Tsallis and Stariolo in [45]. As the name implies, this SA implementation is a generalization of other forms of SA, specifically CSA and FSA, which can be recovered under certain conditions in the GSA formulation. As with FSA, GSA is essentially SA with a modified visiting distribution. The visiting distribution proposed in [45] is given, for a single dimension, by

$$g_{\mathrm{GSA}}(x|\Theta_{\mathrm{GSA}}) = \left[\left(\frac{q_V - 1}{\pi}\right)^{(D/2)}\right] \left[\frac{\Gamma\left(\frac{1}{q_V-1} + \frac{D-1}{2}\right)}{\Gamma\left(\frac{1}{q_V-1} - \frac{1}{2}\right)}\right] \left[\frac{T_{q_V}^{-\frac{D}{3-q_V}}}{\left(1 + \frac{(q_V-1)(x^2)}{\left(T_{q_V}^{2(3-q_V)}\right)}\right)^{\frac{1}{q_V-1}+\frac{D-1}{2}}}\right], \tag{40}$$

where

$$\Theta_{\mathrm{GSA}} = \{T_{q_V}, q_V, D\}, \tag{41}$$

in which q_V is a free parameter selected by the experimenter, and T_{q_V} is a stochastic process control parameter[1] which may, or may not, change during the execution of the SA algorithm, and D is the number of dimensions of the search problem to which GSA is being applied. The function $\Gamma(\cdot)$ is the Gamma function, which is the functional form of a smooth curve that connects and interpolates between the points (x, y), where x and y are related by the function $y = (x1)!$ at the positive integer values for x. Unlike the Cauchy and Gaussian visiting distributions used in CSA and FSA, the GSA visiting distribution has several free parameters that alter the shape and scale of the distribution. The GSA visiting distribution should therefore be conceptualized as a family of related distributions, from which one may be selected to construct a GSA neighborhood function.

Fig. 4 contains a comparison between the distributions produced by several variations of q_V and T_{q_V}, where $D = 1$. Values of q_V near 1 yield very light-tailed distributions, similar to Gaussian distributions. As $q_V \to 1$, $g_{\mathrm{GSA}}(x)$ recovers $g_G(x)$. Similarly, as $q_V \to 2$, $g_{\mathrm{GSA}}(x)$ exactly recovers the Cauchy distribution, $g_C(x)$. Values

[1] This temperature parameter is completely independent from the annealing temperature parameter of SA. The two parameters can vary independently of one another, but will both be annealed over the course of the algorithm.

of q_V greater than 2 do not have a independent analog distribution and have been experimentally shown [46, 48, 47, 49, 51] to yield more efficient SA algorithms, when used as a visiting distribution.

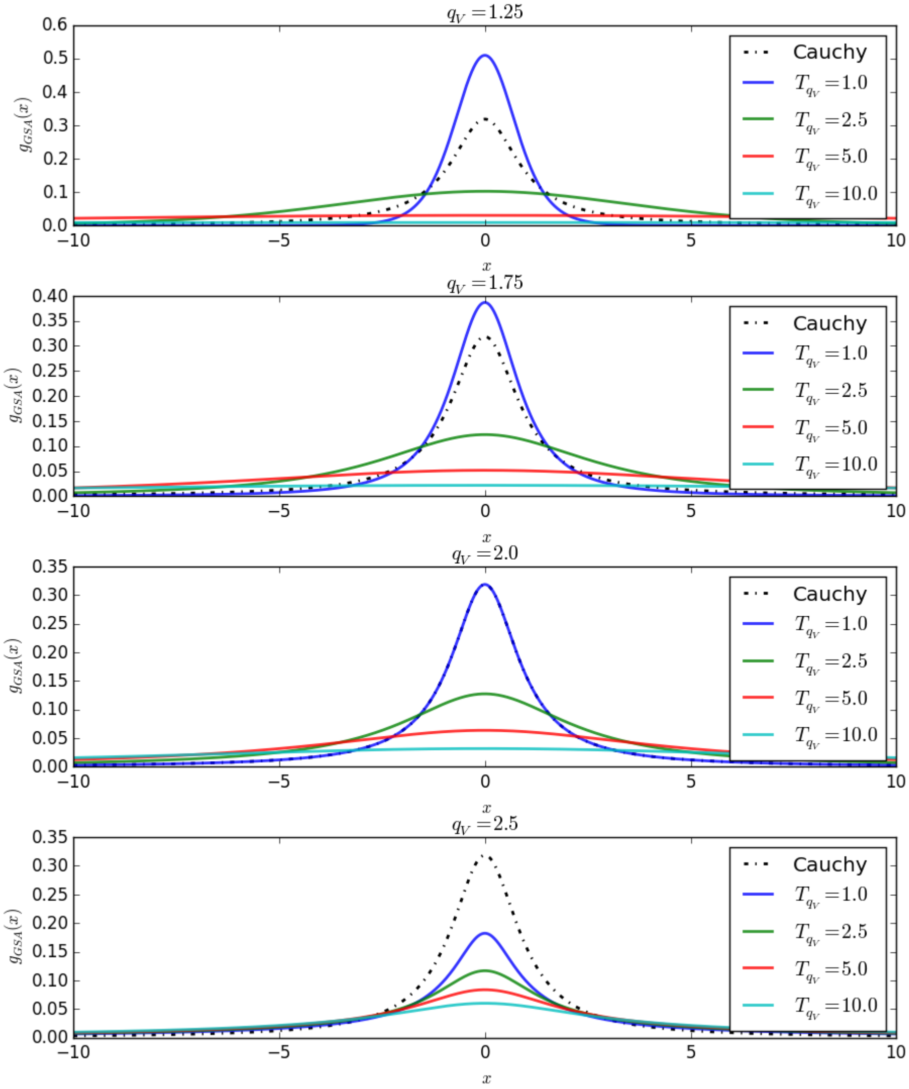


Figure 4. This figure comprises four plots, each of which displays a GSA distribution at various values of T_{q_V}, in order to illustrate the behavior of the GSA distribution for several values of q_V. This figure shows the GSA near the mean, which illustrates the effect of q_v and T_{q_V} on the near-mean domain values, while neglecting the effects on the tails of the distribution.

Fig. 5 displays the same data as Fig. 4, but shows the data over a different scale in order to highlight the impact of the choice of distribution parameters on the tails of the produced distribution. Several trends are seen through the examination of Fig. 4 and Fig. 5. Fig. 4 displays the behavior the GSA distribution near the origin, where, for most combinations of q_V and T_{q_V}, the majority of the probability mass is concentrated. Fig. 5 details the behavior of the GSA distribution at domain values far from the origin, or, in the tails of the distribution. As is shown in Fig. 4, q_V primarily influences the shape of the distribution. Values of q_V near 1 produce distributions with small variance, while larger values of q_V result in distributions with large variance. This behavior is particularly clear in Fig. 5, which shows, in detail, the tails of the distribution. Examining Fig. 5, it is clear that increasing q_V corresponds to an increase in the variance of the resulting distribution. A larger q_V value also corresponds to less tail-behavior sensitivity to the temperature parameter, T_{q_V}. T_{q_V} also influences the distribution shape. As the value of T_{q_V} is increased, the distribution becomes more uniform over the domain. This has important consequences for the application of GSA distribution to stochastic search problems such as SA.

As discussed in [51, 46] the distributions produced by Eq. (40) have several useful properties when used in stochastic search procedures. The longer tails of the GSA distribution when $q_V > 1$ enable more homogeneous visitation of the entire solution space of the problem, relative to a Cauchy distribution. Furthermore, the fact that q_V is selected by the experimenter creates an opportunity for problem-specific construction of the visiting distribution used in the SA implementation. The newly-introduced temperature parameter, T_{q_V}, may also be exploited to escape cost surface local minima. Regardless of the value of q_V, large values of T_{q_V} produce distributions which have high variance, and are therefore able to produce problem configurations which very different from the current state.

While the GSA distribution produces a neighborhood function with several advantageous search characteristics, it does have a significant drawback. The integral

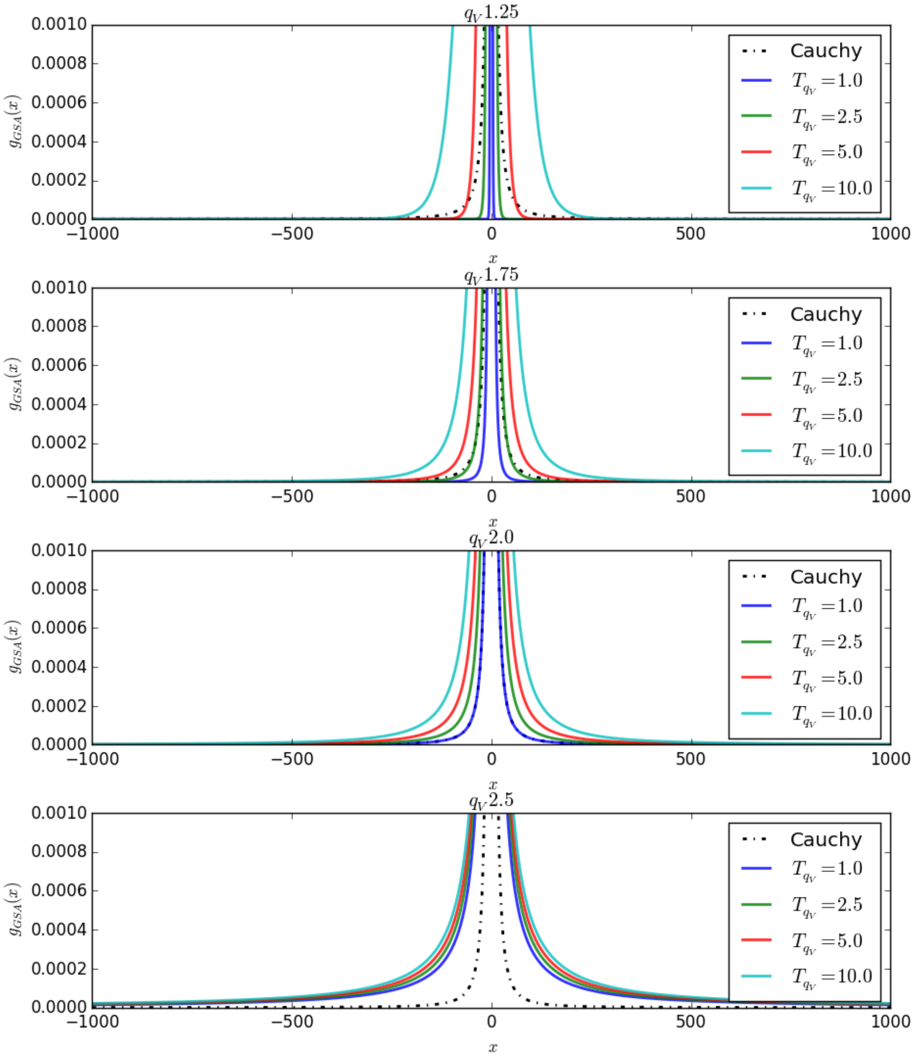


Figure 5. This figure comprises four plots, each of which displays a GSA distribution at various values of T_{q_V}, in order to illustrate the behavior of the GSA distribution for several values of q_V. This figure shows the GSA distribution for domain values far from the mean, which illustrates the effect of q_v and T_{q_V} on the tails of the distribution.

of Eq. (40) has no closed-form analytic solution. The indefinite integral can be constructed, but this operation yields the hypergeometric function, which can be computed as a power series. However, as shown in [46], this approach is computationally expensive. Thus, Eq. (40) is unsuitable for transform-based random sampling. In order to overcome this limitation, a distribution sample caching method is used. This method works by numerically approximating the integral, which yields a series of domain-range value pairs which constitutes an approximation to the cumulative distribution function. To sample the distribution, a number is selected with uniform probability on the interval $[0, 1]$. The numeric integration value which has the minimum-magnitude difference with the randomly selected value is identified, and the displacement value to which it corresponds is returned. This procedure is the numerical equivalent of inverse transform sampling.

An arbitrarily-large set of samples can be constructed using this method. If the sample set is sufficiently large, a random choice from the sample set is approximately equivalent to sampling the original distribution. A large set of samples for each combination of q_V, T_{q_v}, and D can be stored for later access, thus enabling fast numerically-approximate sampling of Eq. (40). A thorough search of the publicly-available resources indicates that there are few systems available for the generation of GSA random variables, despite the popularity and utility of GSA. This thesis develops a system which quickly produces samples from the GSA distribution.

Uniform Visiting Distributions.

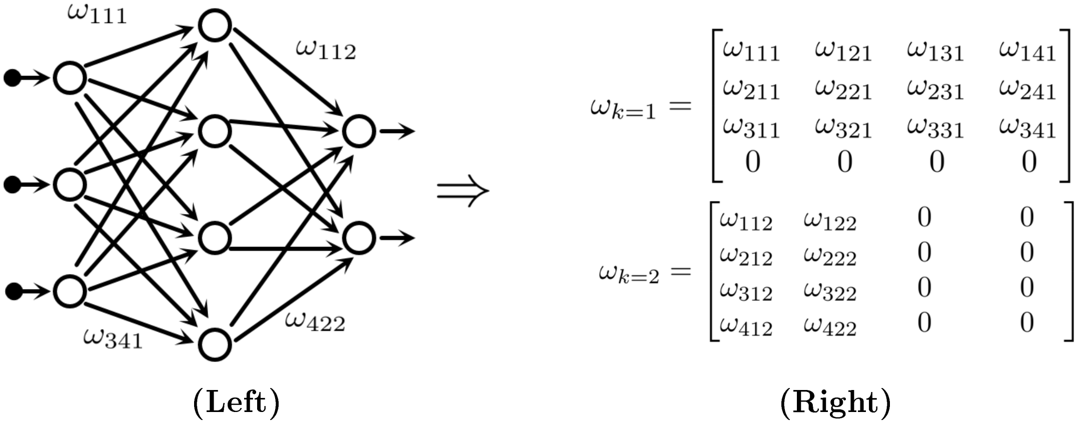
As illustrated in Fig. 2 the probability of observing a particle beyond a classically-impenetrable potential barrier is uniform beyond the barrier[2]. An analogous visiting distribution can be constructed, which models all configuration space movement distances as equally likely. The utility of this visiting distribution is that it makes the entire cost surface accessible each time the neighborhood function is applied to the current state; this property is useful, but prevents any local gradient descent. As such, the uniform visiting distribution serves an upper bound for the trade-space between global and local search policies.

[2] This observation only holds for potential energy surfaces containing a single barrier. The analogous cost surface over neural network weight space is likely to have many barriers corresponding to the superimposed convex spaces around competing conventions of weight configurations which encode similar functions. Thus the analogy used here is only an approximation of the behavior of quantum systems.

Anisotropicity Policies.

In previous work, the visiting distribution is applied isotropically over the free parameters, or dimensions, of the solution space [45, 46]. In the context of SA, isotropic application of the visiting distribution means the next state for each free parameter is a sample from a common, identical distribution. In the SA related works reviewed in this thesis, the visiting distribution is applied isotropically, either explicitly or implicitly. Because all possible realizations of anisotropicity are problem-specific, a detailed discussion of the anisotropicity policies explored in this thesis is deferred to Section 3.3.

3.2 Feed-Forward Neural Network Representation

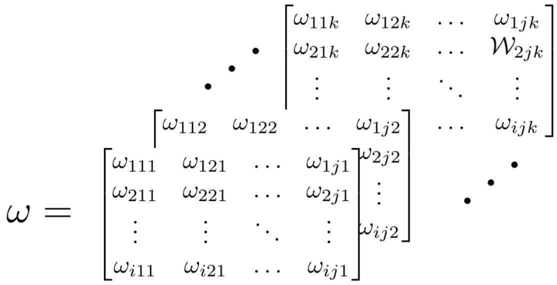


$$\omega_{k=1} = \begin{bmatrix} \omega_{111} & \omega_{121} & \omega_{131} & \omega_{141} \\ \omega_{211} & \omega_{221} & \omega_{231} & \omega_{241} \\ \omega_{311} & \omega_{321} & \omega_{331} & \omega_{341} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\omega_{k=2} = \begin{bmatrix} \omega_{112} & \omega_{122} & 0 & 0 \\ \omega_{212} & \omega_{222} & 0 & 0 \\ \omega_{312} & \omega_{322} & 0 & 0 \\ \omega_{412} & \omega_{422} & 0 & 0 \end{bmatrix}$$

Figure 6. (Left) An arbitrary feed-forward ANN. (Right) The weight matrix representation of the feed-forward ANN in (Left).

The problem of feed-forward neural network (FFNN) synaptic weight selection must be formulated as a combinatorial optimization problem before any formulation of SA can be applied to it. Each synaptic weight in a FFNN may be encoded as a real-valued element in a 3-dimensional relation matrix, denoted as ω_{ijk}. In this encoding scheme, for a given layer, k, of the matrix the row and column indexes indicate the presynaptic and post-synaptic neurons, respectively. The absence of a synaptic connection is indicated by a value of 0 in the matrix element corresponding to that synaptic connection. A nonexistent synapse can be caused by the absence of either the presynaptic or post-synaptic neuron, or by the absence of a connection between the neurons. This weight encoding scheme is depicted graphically in Fig. 6. The weight matrix, ω, therefore encodes a configuration in the problems solution space, and can be visualized as:


$\omega =$
ω_{11k} ω_{12k} $\dots$ ω_{1jk}
ω_{21k} ω_{22k} $\dots$ $\mathcal{W}_{2jk}$
$\vdots$ $\vdots$ $\ddots$ $\vdots$
$\dots$ ω_{ijk}
ω_{112} ω_{122} $\dots$ ω_{1j2}
ω_{2j2}
ω_{ij2}
ω_{111} ω_{121} $\dots$ ω_{1j1}
ω_{211} ω_{221} $\dots$ ω_{2j1}
$\vdots$ $\vdots$ $\ddots$ $\vdots$
ω_{i11} ω_{i21} $\dots$ ω_{ij1}

The total solution space, $\boldsymbol{\mathcal{S}}$, may then be defined as the set of all possible configurations of ω for a given neural network. In the terminology common to the physics literature, ω is the the phase space of the FFNN system. In the synaptic weight selection problem domain, it is more evocative to call $\boldsymbol{\mathcal{S}}$ the weight space of the network, so this convention is adopted throughout this thesis.

A network representation has now been specified, now a formal description of the data on which the network is to operate needs constructed. Feed forward neural networks associate a set of numeric input data elements with a set of numeric output data elements. Thus the formalism which used to represent the data should capture this directional association. In this thesis, a set of ordered pairs is used to represent that association. Formally, a set X is defined such that each element $x \in X$ is an ordered pair of the form

$$x = (\chi, \lambda) = \{\chi, \{\chi, \lambda\}\}$$

where χ and λ are independent sets of real numbers. In this formalism χ is the input data, and λ is the desired output data. X is therefore a specific set of ordered pairs mapping input data to output data. Extending this notion, the notation $\boldsymbol{X}$ is introduced to represent the infinite set of all possible realizations of X, which is all possible data sets which could be presented to an ANN.

In the interest of concise notation, a function $\varphi_L(\omega, \chi)$, which represents set of numeric data produced by propagating the set χ through a FFNN with weights specified by ω, is introduced. The definition, origin, and formal construction of this function is provided in Appendix A. Unless otherwise specified, all neural networks discussed in this thesis use a hyperbolic tangent activation function.

Given the weight space formalism $\boldsymbol{\mathcal{S}}$, and $\boldsymbol{X}$ for the set of all possible data sets, we define a cost function $\mathcal{C}$ as a mapping

$$\mathcal{C} : (\boldsymbol{\mathcal{S}}, \boldsymbol{X}) \to \mathbb{R}.$$

For every data set X, each possible synaptic weight configuration, ω, then corresponds to some cost value $\mathcal{C}(\omega, X)$. Thus, $\mathcal{C}(\omega, X)$ defines a cost surface embedded in the weight space. The objective is now to find a synaptic weight configuration, ω_{opt}, such that

$$\mathcal{C}(\omega_{opt}, X) \leq \mathcal{C}\left(\omega, X\right), \forall \left(\omega \in \boldsymbol{\mathcal{S}}\right).$$

With this framework in place, SA can be applied to transition from a randomly-selected initial state, ω_0, to ω_{opt}, where $\omega_0 \overset{i.i.d.}{\sim} U[-0.1, 0.1]$.

3.3 Applying Simulated Annealing to Feed-Forward Neural Network Weight Modification

The formulations of SA described in Section 3.1 may be applied to any properly formulated combinatorial optimization problem. The representation formalism for the weight parameters of a feed forward neural network presented in Section 3.2 serve as the parameter space in a combinatorial optimization function, and thus enables the application of SA to the problem of neural network weight selection. With a more concrete problem definition in place, it is now possible to precisely specify several SA neighborhood functions for the problem of neural network weight selection. In the following sections, several complete definitions of neighborhood functions are presented for the application of SA to FFNNs. The term *synaptic annealing* is introduced to represent any artificial neural network training algorithm which modifies synaptic weights using SA.

Synaptic Annealing Neighborhood Functions.

A generic synaptic annealing neighborhood function which, given a weight state ω, returns another weight state, ω', which is some modification of the original weight state. The synaptic annealing neighborhood function is generically defined as

$$\mathcal{N}(\omega) = \omega + \alpha(\boldsymbol{G} + \boldsymbol{\mathcal{A}}) \tag{42}$$

where α is the learn rate parameter, $\boldsymbol{G}$ is the neighborhood sample matrix, and $\boldsymbol{\mathcal{A}}$ is the anisotropicity matrix. Each element in the neighborhood sample matrix, $\boldsymbol{G}$, is a sample generated using the random variable generation function $G^{-1}(U)$, where G^{-1} is the sample generation function for a visiting distribution $g(x)$, and U is a uniform random variable over the range $[0, 1]$. Generally, G^{-1} is the inverse transform of the cumulative distribution function of the visiting distribution, or a numeric approximation. Both $\boldsymbol{G}$ and $\boldsymbol{\mathcal{A}}$ have dimensionality equal to that of ω. Additionally, the following constraint is imposed on the matrices:

$$\forall\, \omega_{i,j,k} \in \omega, [\omega_{i,j,k} = 0] \Rightarrow [\boldsymbol{\mathcal{A}}_{i,j,k} = 0] \wedge [\boldsymbol{G}_{i,j,k} = 0]. \tag{43}$$

The constraint specified in Eq. (43) ensures that synaptic connections that are not specified to exist, and are therefore set to exactly 0, are not inadvertently created by the neighborhood function.[3] The term $\alpha(\boldsymbol{G} + \boldsymbol{\mathcal{A}})$ in Eq. (42) is added to the current weight matrix to produce a new weight matrix, and can therefore be thought of as a perturbation of the current state. In the following sections several realizations of this general form, each corresponding to a different form of SA, are presented.

Gaussian (CSA) Synaptic Annealing Neighborhood.

The original description of SA employed a Gaussian visiting distribution [38]. The Gaussian neighborhood function for FFNNs, $\mathcal{N}_G$, is defined as

$$\mathcal{N}_G(\omega) = \omega + \alpha(\boldsymbol{G}_G + \boldsymbol{\mathcal{A}}) \tag{44}$$

where α is the learning rate, $\boldsymbol{G}_G$ a matrix of samples drawn from the standard normal distribution such that

$$\boldsymbol{G}_G \overset{iid}{\sim} N(0, 1), \tag{45}$$

[3] Strictly speaking, it is possible for a weight to be set to 0 in the normal operation of the SA algorithm. If this were to occur, the constraint specified in Eq. (43) would prevent that weight from ever being modified again. This scenario was never observed in the course of this work, and is very unlikely, but is not explicitly forbidden by the implementation of synaptic annealing proposed in this work.

and $\boldsymbol{\mathcal{A}}$ is an anisotropicity matrix. The neighborhood function given in Eq. (44) is an application of the canonical form of simulated annealing to the problem of selecting a weight configuration for a FFNN. The term classical is used here because the underlying simulated annealing model can be described entirely in terms of classical statistical mechanics. To interpret this in terms of the analogy present in Sec. 3.1, the probability of the Gaussian visiting distribution used in CSA generating a weight space distance large enough to transition the system across a large energy barrier is effectively 0. In the following sections, models which approximate quantum mechanical phenomena are constructed.

Cauchy (FSA) Synaptic Annealing Neighborhood.

The Cauchy neighborhood function used for synaptic annealing, $\mathcal{N}_C$, is given by

$$\mathcal{N}_C(\omega) = \omega + \alpha(\boldsymbol{G}_C + \boldsymbol{\mathcal{A}}) \tag{46}$$

where α is the learning rate, $\boldsymbol{G}_C$ a matrix of samples drawn from the Cauchy distribution such that

$$\boldsymbol{G}_C \overset{iid}{\sim} Cauchy(\Theta = \{0, 1\}), \tag{47}$$

and $\boldsymbol{\mathcal{A}}$ is an anisotropicity matrix. The perturbation of weight configuration produced by Eq. (46) will be sufficiently-large to allow the system to occasionally traverse cost surface barriers, thereby escaping entrapment in local minima.

Tsallis (GSA) Synaptic Annealing Neighborhood.

The GSA neighborhood function used for synaptic annealing $\mathcal{N}_T$ is defined as

$$\mathcal{N}_{\mathrm{GSA}}(\omega) = \omega + \alpha(\boldsymbol{G}_{\mathrm{GSA}} + \boldsymbol{\mathcal{A}}) \tag{48}$$

where α is the learning rate, $\boldsymbol{G}_{\mathrm{GSA}}$ a matrix of samples drawn from the GSA distribution such that

$$\boldsymbol{G}_{\mathrm{GSA}} \overset{iid}{\sim} GSA(\Theta = \{q_V, T_{q_V}, D\}), \tag{49}$$

and $\boldsymbol{\mathcal{A}}$ is an anisotropicity matrix. Relative to the weight space traversal jumps made by an annealing system which has a Cauchy neighborhood function, a system that uses the GSA neighborhood function will result in longer jumps that occur more frequently, thereby ensuring a more homogeneous search of the weight space.

Uniform Synaptic Annealing Neighborhood.

Finally, the uniform neighborhood function used for synaptic annealing ($\mathcal{N}_U$), defined as

$$\mathcal{N}_U(\omega) = \omega + \alpha(\boldsymbol{G}_U + \boldsymbol{\mathcal{A}}) \tag{50}$$

where α is the learning rate, $\boldsymbol{G}_U$ a matrix of samples drawn from the uniform distribution such that

$$\boldsymbol{G}_U \overset{iid}{\sim} U(-0.5, 0.5), \tag{51}$$

and $\boldsymbol{\mathcal{A}}$ is an anisotropicity matrix.

Feed-Forward Neural Network Anisotropicity Policies.

In the course of developing the synaptic weight selection system presented in this thesis, it was observed that it is sometimes advantageous to perturb different synaptic weights using different distributions during training. Since each synaptic weight represents a free parameter in the specification of a solution, the term anisotropicity, meaning the application of some effect differently on a different free parameter, is adopted. There are several possible realization of anisotropicity for the neural network weight selection problem, two of which are considered in this work. In this thesis, anisotropicity is specified as a modification of the perturbation.

Isotropic (Null) Anisotropicity.

For completeness, a default anisotropicity is specified. The isotropic, or null, anisotropicity perturbation modification is simply a 0 matrix of the same dimensionality as the perturbation matrix. This matrix, when added to the perturbation matrix produces no change, thereby leaving the originally-isotropic application of the visiting distribution isotropic.

Weight-Based Anisotropicity.

When evaluating the traversal characteristics of the synaptic annealing algorithms constructed in the this thesis, it is observed that the sum of squared weight values grew considerably during training. Previous work in [11] has show that large weight magnitudes result in networks with high bias, and correspondingly high generalization error. It is intuitively clear that, when using a sigmoidal activation function, a very high-magnitude weight value effectively converts the afferent neuron, relative to that synapse, into a bias. Unless an equally large-magnitude weight of the opposite sign is introduced, or the weight value fluctuates to a smaller value, the afferent neuron will remain a bias. One of these contingencies may occur, but it is unlikely that such an event would reduce the total training error of the network, and is therefore likely to be rejected. In [11], Lee, et al. propose a method called multiobjective hybrid greedy simulated annealing (MOHGSA), which uses SA to minimize both the cost function *and* the sum of squared weights. By doing so, Lee, et al. were able to reduce the the generalization error of a neural network trained by SA. The MOHGSA approach works by exerting a selection pressure that lowers the likelihood of the SA algorithm accepting moves that reduce the cost function if the move also increases the sum of squared weights. An analogous procedure, which is a different means to the same end, is accomplished using anisotropicity in the visiting distribution, with respect to the *magnitude* of the weight. This anisotropicity policy is called weight-based anisotropicity, or weight anisotropicity. The weight anisotropicity matrix is given by

$$\boldsymbol{\mathcal{A}}_{i,j,k} = -(|\omega_{i,j,k}| > t_a)\frac{\omega_{i,j,k}}{s_a},\ \forall\ \omega_{i,j,k} \in \omega, \tag{52}$$

where $(|\omega_{i,j,k}| > 1)$ is 1 if true and 0 if false, t_a is a threshold parameter where $t_A \geq 0$, and s_a is a shift parameter where $s_a \geq 1$. The threshold parameter t_a determines the weight magnitude at which the anisotropic effects occur. If $|\omega_{i,j,k}| < t_a$, then $\boldsymbol{\mathcal{A}}_{i,j,k} = 0$. If the threshold is exceeded, then a non-zero value is subtracted from the perturbation matrix element. Note that this subtraction corresponds to shifting the mean of the visiting distribution, which is by default the current weight value, thereby creating a new effective visiting distribution that is different for each weight in the network, and hence is anisotropic. The value subtracted from the mean is given by $\omega_{i,j,k}/s_a$, which is positive when $\omega_{i,j,k}$ is positive, and negative when $\omega_{i,j,k}$ is negative. Thus, whenever the threshold weight value is exceeded, the mean of the effective visiting distribution is brought closer to zero. The amount by which the mean is brought closer to 0 is determined by s_a. For example, if $s_a = 2$, then the mean of the effective visiting distribution will be brought halfway back to 0. In the limit as $s_a \to \infty$, the anisotropicity has no effect, whereas in the $s_a \to 1$ limit, the anisotropicity yield an effective visiting distribution with mean 0. This has the effect of allowing unconstrained exploration of the weight space inside the bounds of the threshold, while reducing the probability of extended weight space traversal outside of the bounds of t_a. The weight anisotropicity thereby prevents unconstrained traversal into large weight values, which in turn reduces the number of bias-like neurons created. This reduction in bias-like neurons should prevent divergence between the training and validation errors.

3.4 Cost Functions

All variations of synaptic annealing require the specification of a heuristic cost function, which conforms to the specification given in Section 3.2. In this thesis, the synaptic annealing algorithm is applied to two broad problem classes: regression and classification. To apply synaptic annealing to both of these problem classes, two cost functions are constructed.

Regression Error Cost Function.

The cost function chosen to serve as a heuristic of regression error

$$\mathcal{C}_r(\omega, X) = \sum_{(\chi,\lambda)\in X} [\lambda - \varphi(\omega, \chi)]^{\circ 2} \tag{53}$$

where ω is a weight matrix, φ is the propagation function, and $\mathcal{X}$ is the set of ordered pairs mapping a set of input data χ to a set of output values λ, in which χ is the first entry and λ is the second entry. This cost function yields the squared error (SE) between the desired output set λ and the output of the forward propagation of the input data, $\varphi(\omega, \chi)$.

Classification Error Cost Function.

The classification cost function, $\mathcal{C}_c$, yields the number of *incorrectly* classified observations in a given data set, which is

$$\mathcal{C}_c(\omega, X) = \sum_{(\chi,\lambda)\in X} \left(\lambda \neq \delta(\varphi(\omega, \chi))\right). \tag{54}$$

where ω is a weight matrix, φ is the propagation function, and X is the set of ordered pairs mapping a set of input data χ to a set of output values λ, in which χ is the first entry and λ is the second entry, and δ is a function which returns which maps any vector a to another vector b where

$$\forall b_i \in b, b_i = \begin{cases} 1, & a_i = \max(a), \\ 0, & a_i \neq \max(a). \end{cases}$$

This function is used to evaluate the performance of a network configuration, for a given set of data, when that data consists of categorical, or labeled, data. Though the classification cost function will always be used to report the performance of a network configuration when applied to classification data, it may or may not be used as the cost function which informs the synaptic annealing algorithm. Often, it is advantageous to train a neural network using the regression error function [10] presented in Section 3.4, and then report the performance of that network using the classification error.

3.5 Synaptic Annealing Algorithm Specification

In the preceding sections all of the required components for a complete specification of the synaptic annealing algorithm have been defined. With these definitions in place, the algorithmic description of the synaptic annealing algorithm is now specified, and presented in Algorithm 2.

Algorithm 2 This algorithm describes a metaheuristic method for selecting the synaptic weights of a FFNN such that a cost function is minimized. *Inputs:* ω_0, a weight matrix specifying the synaptic weights for a neural network; this variable is generally, but not necessarily, randomly generated. X, a data set. $\mathcal{C}$, a cost function defined on a data set and weight matrix. $T(\cdot)$, a temperature schedule function, as described in Section 2.2, which returns the temperature value given the number of elapsed epochs, t. ϵ, a minimum temperature value which, once attained, causes the algorithm to halt; values of ϵ less than or equal to 0.0 may prevent the algorithm from halting, and are therefore invalid inputs. $\boldsymbol{G}$, a random-matrix production function which produces matrices according to some specified distribution. $\boldsymbol{\mathcal{A}}$, a random-matrix production function which produces matrices according to some specified anisotropicity function. *Outputs:* ω_{opt}, a weight matrix specifying the weight state which corresponds to the minimum cost function value, given the input parameters.

```
procedure SynapticAnnealing($\omega_0$, $X$, $\mathcal{C}$, $\epsilon$, $\boldsymbol{G}$, $\boldsymbol{\mathcal{A}}$)
    $\omega \leftarrow \omega_0$                                   ▷ Initialize the weight matrix.
    $t \leftarrow 1$                                               ▷ Initialize the time.
    $c \leftarrow (\mathcal{C}(\omega, X))$                        ▷ Initialize the cost.
    while $T(t) > \epsilon$ do        ▷ Iterate until the temperature schedule is less than $\epsilon$.
        $\omega' \leftarrow \omega + \alpha(\boldsymbol{G} + \boldsymbol{\mathcal{A}})$ ▷ Apply the neighborhood function by perturbing $\omega$.
        $c' \leftarrow \mathcal{C}(\omega', X)$                    ▷ Compute the cost of $\omega'$ on the data set $X$.
        $\Delta C \leftarrow (c - c')$                              ▷ Compute the cost change.
        if $(\Delta C \leq 0) \vee (\exp(\Delta C / T(t)) > U(0,1))$ then    ▷ Apply Metropolis criterion.
            $\omega \leftarrow \omega'$
            $c \leftarrow c'$
        end if
        $t \leftarrow t + 1$                                       ▷ Increment the time.
    end while
    $\omega_{opt} \leftarrow \omega$                               ▷ Declare the optimal solution.
    return $(\omega_{opt})$                                        ▷ Return the optimal solution.
end procedure
```

3.6 Weight Space Traversal

In SA, the purpose of the neighborhood function is to control the traversal of the solution space of the problem. For synaptic annealing, the solution space is the synaptic weight space, thus synaptic annealing neighborhood functions control the

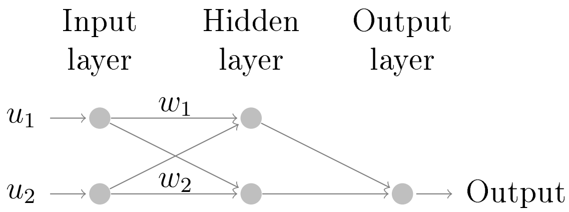


Figure 7. The feed-forward neural network used in the weight space traversal evaluations.

traversal of the weight space. The performance of a neighborhood function is entirely dictated by the weight space traversal, or walk, that it produces. In order to characterize the traversal properties of each neighborhood function, a traversal through a two-dimensional subset of the weight space is analyzed for each neighborhood function. A small FFNN, with two input neurons, two hidden layer neurons, and a single output layer neuron, is trained using synaptic annealing to solve a simple functional approximation problem. This network is depicted in Fig. 7. The network uses a $\tanh(\cdot)$ activation function, an initial annealing temperature of 10, and an initial learn rate of $\alpha = 0.001$.

The test function is the complex-interaction function used in [11], which is given by

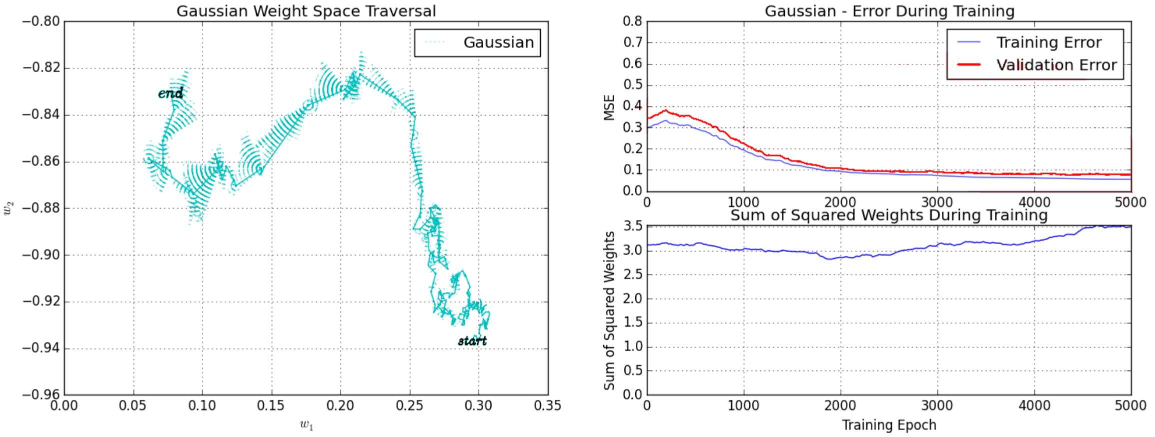
$$f_{\mathrm{CI}}(u_1, u_2) = 1.9(1.35 + e^{0.5(u_1+1)} \sin(13(0.5u_1 - 0.1)^2) e^{0.5(u_2+1)} \sin(3.5u_2 + 3.5)). \quad (55)$$

This function is used because it allows for comparison with published results, and has several features that make it useful when studying generalization error. In this section, f_{CI} is used for demonstrative purposes only; a detailed description of the function can be found in Section 4.1. The synaptic annealing algorithm attempts to minimize the squared error (SE) cost function, as defined in Section 3.4, on 100 randomly-drawn samples from f_{CI}. The validation error of the synaptic annealing algorithm is also evaluated using an additional 100 randomly drawn samples, which are not presented to the algorithm during training. The same validation and training samples are used for the analysis of the weight space traversal produced by each neighborhood function. This section contains an analysis of the weight space traversal properties of synaptic annealing during a relatively short training period. A more thorough analysis of the performance of synaptic annealing on this problem is presented in Chapter IV. For each neighborhood function, the traversal through the $\omega_{1,1,1}$-$\omega_{2,2,1}$ space is analyzed. $\omega_{1,1,1}$ is the synaptic weight from input neuron 1 to hidden neuron 1, while $\omega_{2,2,1}$ is the synaptic weight from input neuron 2 to hidden layer neuron 2. The weight space traversal behavior of the algorithm is empirically found to be independent of the specific choice of weights. In the interest of expedience and conciseness, in this section a traversal produced by a synaptic annealing algorithm using a neighborhood function for which the visiting distribution is x, will be called a *x traversal.*

Gaussian Neighborhood Function Weight Space Traversal.

The weight space traversal exhibited by the Gaussian visiting distribution is consistent with a Gaussian random walk, as shown in Fig. 8 (Left). A single exemplary traversal is shown in Fig. 8; while the figure contains only one traversal, several identical experiments are conducted to ensure that the exemplar shown in the figure, as well as in Fig. 9, Fig. 10, Fig. 11, and Fig. 12, are typical of the execution of the algorithm.

Fig. 8 (Left) indicates that the traversal begins near the origin of the weight space and proceeds outward. The initial location is dictated by the values to which the weights in the FFNN are initialized, which are always random and small in magnitude in this work. The traversal proceeds to higher-magnitude weights as it follows the local cost surface of the training data set. Initially, most moves are accepted, indicated by a solid line in the figure, because the temperature is high during the first several



(Left) **(Right)**

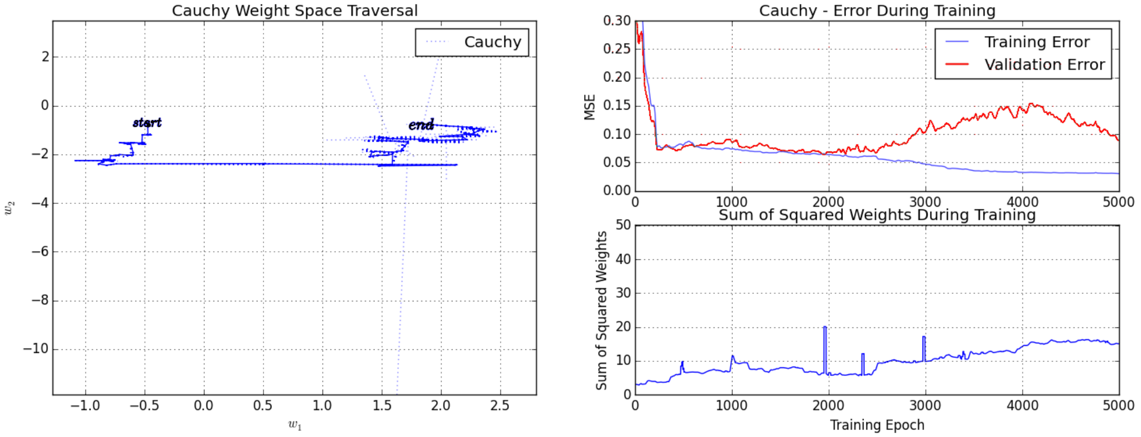
Figure 8. (Left) A plot showing the traversal of the w_1, w_2 subspace of the weight space over the course of $5,000$ epochs produced by a synaptic annealing algorithm employing a Gaussian visiting distribution. A solid line indicates a move which was accepted by the simulate annealing algorithm, while a dashed line indicates a move which was rejected. In this figure, only a few rejected moves are visible. The word *start* indicates the initial value of (w_1, w_2), while the word *end* denotes the final value. (Right-Top) A plot showing the both the training and validation MSE of the results produced by the neural network in each epoch, smoothed using a central moving window average with a width of 21. This is the post-perturbation error, meaning that the error associated with moves that were rejected is shown. (Right-Bottom) A plot showing the sum of squared weights of the neural network during each training epoch.

training epochs. As the traversal proceeds and the temperature is lowered, the number of rejected moves, indicated by a dotted line, increases. The algorithm settles in a local minima at $w_1 = 0.075$ $w_2 = -0.83$. Examining Fig. 8 (Right), a clear downward trend in training error exists, and upward trend in sum of squared weights is present. It is also clear that there is no trend in the validation error after an initial period of decline. Because large synaptic weights tend to cause their afferent neurons to behave like biases, these trends are interpreted as the network memorizing the training data through the construction of fine-tuned biases. Thus, given sufficient time and sufficiently-many free parameters, the training error will become arbitrary low, while the validation error will remain relatively large.

Cauchy Neighborhood Function Weight Space Traversal.

A neighborhood function based on a Cauchy visiting distribution produces a very different traversal pattern than the Gaussian visiting distribution. Fig. 9 (Left) shows that a Cauchy visiting distribution causes a traversal that is prone to very long jumps through single dimensions in the weight space; note the difference in scale on the w_1 axis, relative to that of the traversal plot displayed in Fig. 8 (Left). This long-jump exploration behavior can be explained as a consequence of the shape of the Cauchy distribution. As can be seen in Fig. 4 and Fig. 5, the tails of the Cauchy distribution are longer than those of the Gaussian distribution. This tail characteristic translates to a low, but non-zero probability of producing a long jump in a given dimension, thus enabling long jumps. However, because the probability of a long jump is low, it is unlikely to occur in both of the examined dimensions simultaneously, thereby producing the distinctive single-dimensional search pattern exhibited by the Cauchy visiting distribution. A closer inspection of Fig. 9 (Left) reveals that the Cauchy visiting distribution produces a traversal which is similar to that of Gaussian distribution when the generated traversal distances remain near the mean.

One can see how the traversal characteristics of a Cauchy visiting distribution effect the performance of a synaptic annealing algorithm by examining Fig. 9 (Right). Unlike the results generated by the Gaussian visiting distribution, the results in the upper plot of Fig. 9 (Right) show a downward trend in training error. The validation error initially follows the downward trend in training error, but begins to increase around epoch 2500. This increase in validation error, while the training error decreases, is evidence of overfitting. A complication associated with the use of a Cauchy visiting distribution for synaptic annealing is visible in Fig. 9 (Right) at approximately epoch 2500. Starting at this epoch, and continuing for the next few epochs, the sum of squared weights increases considerably, indicating that a sig-



(Left) **(Right)**

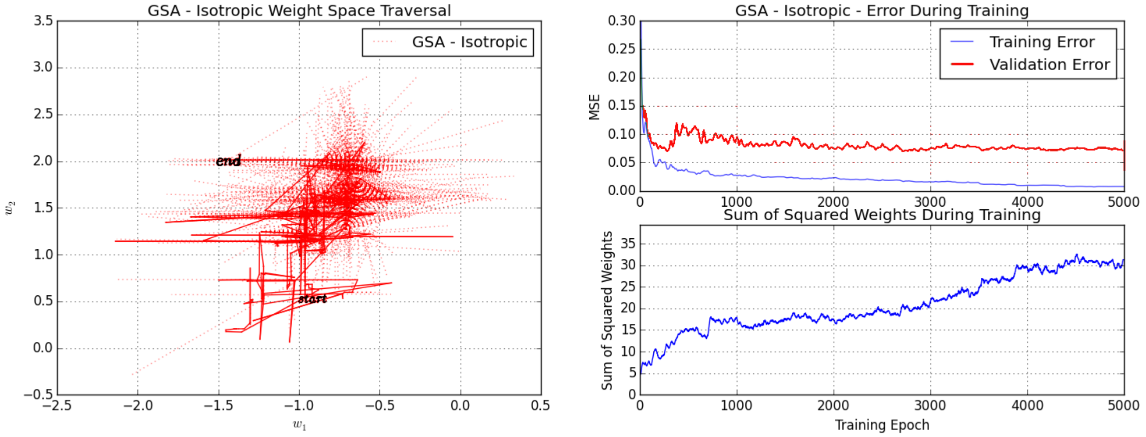
Figure 9. (Left) A plot showing the traversal of the w_1, w_2 subspace of the weight space over the course of $5,000$ epochs produced by a synaptic annealing algorithm employing a Cauchy visiting distribution. A solid line indicates a move which was accepted by the simulate annealing algorithm, while a dashed line indicates a move which was rejected. In this figure, only a few rejected moves are visible. The word *start* indicates the initial value of (w_1, w_2), while the word *end* denotes the final value. (Right-Top) A plot showing the both the training and validation MSE of the results produced by the neural network in each epoch, smoothed using a central moving window average with a width of 21. This is the post-perturbation error, meaning that the error associated with moves that were rejected is shown. (Right-Bottom) A plot showing the sum of squared weights of the neural network during each training epoch.

nificant change in the network occurred. This change decreased the training error marginally, while increasing the validation error. As the sum of squared weights increased throughout the remainder of the training process, the validation error and training error diverged considerably. Near the end of the training process, the training error decreases by about half while the validation error nearly doubles. Thus the data in Fig. 9 provides strong (albeit circumstantial) evidence that the Cauchy visiting distribution results in a biased neural network. Further, there is no reason to conclude that this biasing problem would improve, if given additional training time.

Isotropic GSA Neighborhood Function Weight Space Traversal.

The weight space traversal produced by a neighborhood function using an isotropically applied GSA visiting distribution is displayed in Fig. 10 (Left). This traversal produces a homogeneous pseudo-global search pattern indicative of GSA. As with the Cauchy visiting distribution neighborhood function, the traversal produced by the GSA distribution neighborhood function can be explained as a consequence of the distributions tail characteristics. The GSA distribution used to construct the traversal in Fig. 10 (Left) had the parameters $q_V = 2.5$, T_{q_V}, and $D = 1$. As can be seen in the lower plot of Fig. 4 and Fig. 5, the GSA distribution, using the parameters indicated, has significantly heavier and longer tails than the Cauchy distribution. This translates to a traversal which exhibits jumps that are longer and considerably more frequent, relative to a Cauchy traversal.

Given the observation, made in Section 3.6, that large jumps through the weight space tend to create synaptic weights that cause their afferent neurons to behave as biases, it is expected that the GSA traversal will readily produce such bias-like neurons. Fig. 10 (Right) provides evidence that this prediction is correct. The lower plot of Fig. 10 (Right) indicates that the sum of squared weights created by the GSA traversal surpasses the largest value observed in the Cauchy traversal at epoch 250. It should therefore be expected that a subsequent steep decrease in training error occurs; this is indeed what is observed in the upper plot of Fig. 10 (Right). Further, it should be expected that the validation error will decrease as well, in so far as the training set is representative of the entire data set, but not to the same extent as the training error. Additionally, the validation error should gradually diverge from the training error as the sum of squared weight increases. These predicted characteristics, too, are present in Fig. 10 (Right). In order to reduce the generalization error introduced by these features of the GSA neighborhood function when applied to synaptic annealing,



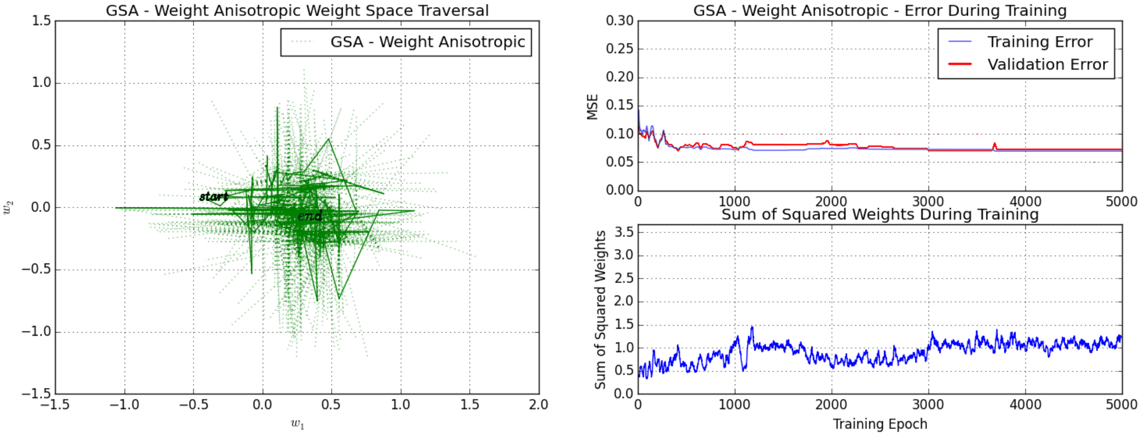
(Left) **(Right)**

Figure 10. (Left) A plot showing the traversal of the w_1, w_2 subspace of the weight space over the course of $5,000$ epochs produced by a synaptic annealing algorithm employing an isotropic GSA visiting distribution. A solid line indicates a move which was accepted by the simulate annealing algorithm, while a dashed line indicates a move which was rejected. In this figure, only a few rejected moves are visible. The word *start* indicates the initial value of (w_1, w_2), while the word *end* denotes the final value. (Right-Top) A plot showing the both the training and validation MSE of the results produced by the neural network in each epoch, smoothed using a central moving window average with a width of 21. This is the post-perturbation error, meaning that the error associated with moves that were rejected is shown. (Right-Bottom) A plot showing the sum of squared weights of the neural network during each training epoch.

an anisotropic modification of the distribution is examined in the following section.

Anisotropic GSA Neighborhood Function Weight Space Traversal.

Fig. 11 (Left) is an example weight space traversal produced by a neighborhood function which uses a GSA distribution with weight anisotropicity. As defined in Section 3.3, the weight anisotropicity is an operator applied to the weight perturbation matrix. For each weight in the network, the operator shifts the mean of the visiting distribution for that weight closer to the origin in proportion to the magnitude of the weight. Thus, the further a synaptic weight is from zero, the more the mean of its visiting distribution will be shifted toward zero. The effect of the weight anisotropicity operator is to reduce the likelihood of weight space jumps that result in a state far from the origin of the weight space. Contrasting Fig. 11 (Left) and Fig. 10 (Left), it is clear that the former depicts a traversal that has the homogeneous search characteristics of a GSA traversal, while simultaneously avoiding protracted traversals into large weight space values.



(Left) **(Right)**

Figure 11. (Left) A plot showing the traversal of the w_1, w_2 subspace of the weight space over the course of $5,000$ epochs produced by a synaptic annealing algorithm employing a GSA visiting distribution with synaptic weight-based anisotropicity. A solid line indicates a move which was accepted by the simulate annealing algorithm, while a dashed line indicates a move which was rejected. In this figure, only a few rejected moves are visible. The word *start* indicates the initial value of (w_1, w_2), while the word *end* denotes the final value. (Right-Top) A plot showing the both the training and validation MSE of the results produced by the neural network in each epoch, smoothed using a central moving window average with a width of 21. This is the post-perturbation error, meaning that the error associated with moves that were rejected is shown. (Right-Bottom) A plot showing the sum of squared weights of the neural network during each training epoch.

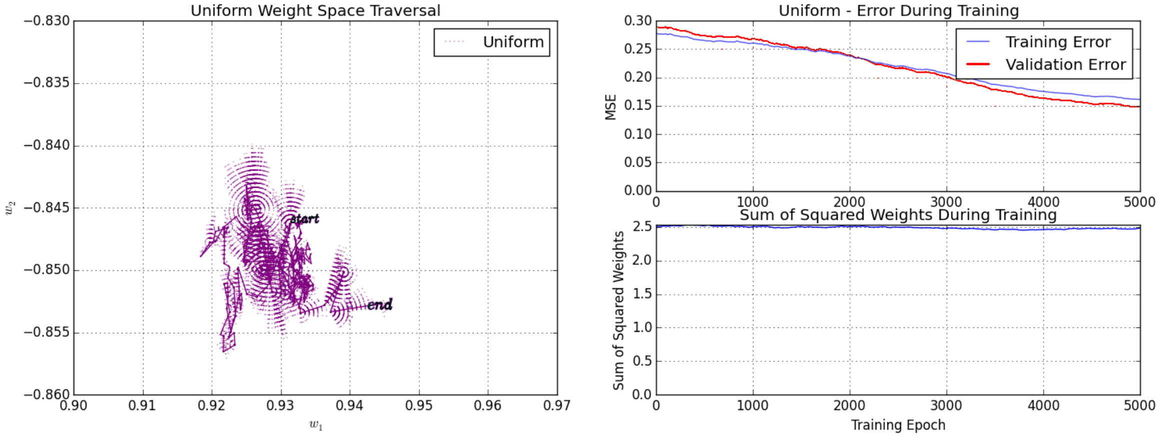
If properly constructed, the weight anisotropicity mechanism prevents weights from becoming large enough to saturate neurons in the network. Theoretically, this is a desirable characteristic for a neighborhood function as it will prevent very large weights from arising in the network, which prevents the creation of saturated neurons. A network with fewer saturated neurons acting as biases will be less biased, by definition. A synaptic annealing algorithm using a weight anisotropic GSA neighborhood function should yield lower validation set errors, relative to an algorithm that uses an isotropic GSA neighborhood function. In the upper plot of Fig. 11 (Right), the validation and training error for the traversal shown in Fig. 11 (Left) are displayed. In contrast to the preceding weight space traversal performance plots, the training and validation error of the weight anisotropic GSA traversal are found to be in close agreement. Additionally, the lower plot of Fig. 11 (Right) indicates that there is no significant trend in the sum of squared weight values for the network, which is also a property unique to the weight anisotropic GSA neighborhood function. These results indicate that weight anisotropicity may serve as an alternative weight minimization technique to the multiobjective simulated annealing approach described in [11].

Uniform Neighborhood Function Weight Space Traversal.

The weight space traversal produced by a uniform neighborhood distribution, shown in Fig. 12 (Left), is very similar to that of a Gaussian neighborhood function. The most significant difference between the traversals is total area of the weight space explored. The Gaussian traversal covers considerably more area than the Uniform traversal. This behavioral difference is the result of the fact that the uniform distribution is over the range $\left[-\frac{1}{2}, \frac{1}{2}\right]$, which is much smaller than the domain of the standard normal distribution.

Examining the training error performance shown in Fig. 12 (Right), it is clear that the exemplar uniform neighborhood function significantly under-performed, with respect to the Gaussian neighborhood function. This deficiency can be explained primarily as a consequence of the fact that the uniform distribution has compact support. The restricted search range combined with the learning rate used in these traversal experiments yield very little change in the state of the network, as illustrated in the sum of squared weights plot displayed in the lower pane of Fig. 12 (Right).

Though the preceding analysis is based only on the support interval $\left[-\frac{1}{2}, \frac{1}{2}\right]$,



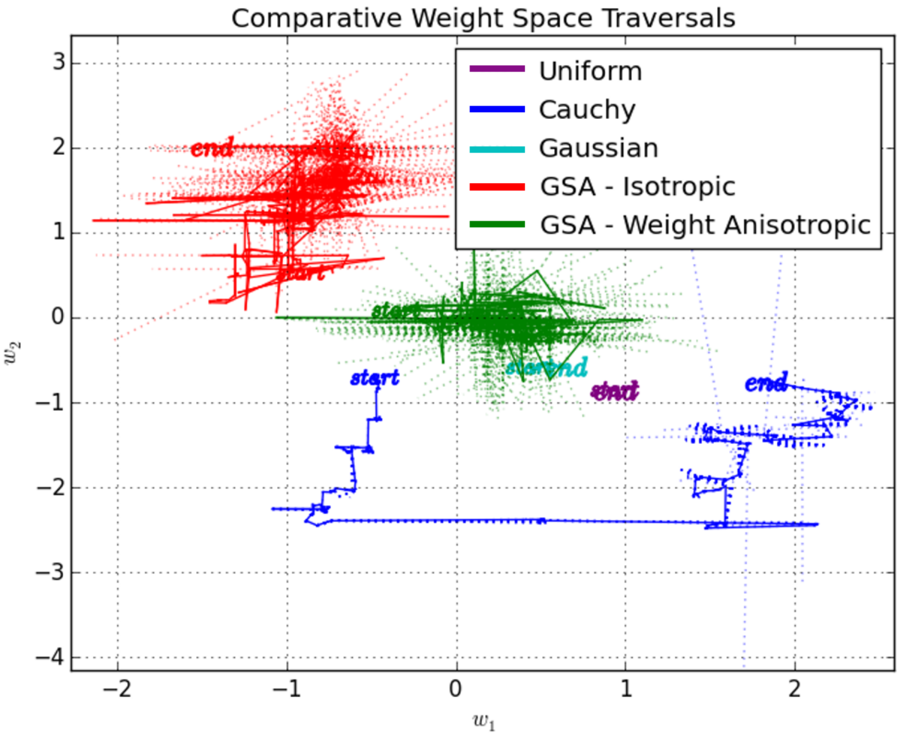
(Left) **(Right)**

Figure 12. (Left) A plot showing the traversal of the (w_1, w_2) subspace of the weight space over the course of $5,000$ epochs produced by a synaptic annealing algorithm employing a visiting distribution which is uniform over the range $\left[-\frac{1}{2}, \frac{1}{2}\right]$. A solid line indicates a move which was accepted by the simulate annealing algorithm, while a dashed line indicates a move which was rejected. In this figure, only a few rejected moves are visible. The word *start* indicates the initial value of (w_1, w_2), while the word *end* denotes the final value. (Right-Top) A plot showing the both the training and validation MSE of the results produced by the neural network in each epoch, smoothed using a central moving window average with a width of 21. This is the post-perturbation error, meaning that the error associated with moves that were rejected is shown. (Right-Bottom) A plot showing the sum of squared weights of the neural network during each training epoch.

myriad support intervals are found to yield similar results. For small intervals, the compact support of the uniform distribution prevents pseudo-local search, thereby reducing the chance of escaping local minima. For large intervals, very little local gradient descent occurs, resulting in a random search of the weight space. Thus, the uniform visiting distribution is empirically found to yield results which are inferior to those of other visiting distributions. As such, the uniform visiting distribution is excluded from analysis in Chapter IV.

Comparing Weight Space Traversal Methods.

It is instructive to directly compare the traversals produced by the various distributions considered in the preceding sections. Fig. 13 displays these traversals on a common weight space; several interesting traversal characteristics are present. The impact of anisotropicity on the traversal characteristics of the synaptic annealing algorithm are demonstrated by the difference in the weight-space location in which the respective searches occur: The anisotropic traversal is limited to a range near the origin, while the isotropic traversal searches a higher magnitude region of the weight space. The relatively diminutive total traversal length of both the Gaussian and uniform neighborhood functions are also apparent when compared to the range of the traversals produced by the other visiting distributions. The regression error performance of the each neighborhood function is displayed in Table 1.



(Left)

Figure 13. (Left) A plot juxtaposing the weight space traversals of produced by a synaptic annealing algorithm employing several different visiting distributions.

Examining Table 1 suggests a few trends. As would be expected, the training set MSE of GSA is considerably lower than that of FSA (Cauchy visiting distribution),

Table 1. Training and Validation Set Mean Squared Error at Epoch $5,000$

Neighborhood Function Visiting Distribution	Training Error	Validation Error
Gaussian	0.05549	0.07743
Cauchy	0.03028	0.08878
Isotropic GSA	0.07390	0.07202
Weight Anisotropic GSA	0.06947	0.07229
Uniform	0.16111	0.14773

which is in turn better than CSA (Gaussian visiting distribution). These findings are consistent with previous findings [46, 45] regarding the efficiency of various SA implementations. Interestingly, the MSE of the validation set, which in this case is specific to the problem of FFNN synaptic weight selection, does not follow this trend. While it may seem that introducing weight anisotropicity only served to increase the final training set MSE of the synaptic annealing algorithm, the mechanism that produced this increase in error is actually advantageous, as discussed in Section 3.6.

IV. Experimental Results and Analysis

This chapter contains a complete description of the design of experiments used to evaluate the effectiveness of the various implementations of synaptic annealing described in Chapter III. Several data sets are employed in these experiments, and each is described in detail; relevant features of the data seats are highlighted. The experimental methodology used is described and justified. Finally, each set of results is analyzed and relations are drawn between the structure of synaptic annealing, and the produced results.

4.1 Design of Experiments

In this section the experimental design used to evaluated the performance of synaptic annealing is described. The section contains descriptions of several data sets, and the properties thereof. The preprocessing procedure, which is applied to each data set before any network training occurs, is detailed and justified. The algorithmic parameters used to perform the experiments using synaptic annealing are specified and discussed.

Data Sets.

The foundation of the experimental framework used to evaluate the synaptic annealing methodology is set of data sets to which the methodology is applied. In order to ensure that the methodology is generally applicable, it is applied to several different data sets which vary in complexity and organization. These data set can be partitioned into two broad classes: classification and regression, each of which is discussed in the following section. Table 2 summarizes some basic properties of each data set.

Table 2. A summary of each of the classification data sets

Data Set Name	Input Vector Dimensionality	Number of Classes	Observations
Wine	13	3	178
Iris	4	3	150
Cancer	30	2	569

Classification Data Sets.

Three data sets were selected to evaluate the effectiveness of synaptic annealing when applied to classification problems. Each data set has a different number of input dimensions, as well as different class distributions; additionally, each data set comes from a separate discipline. All data sets are publicly available for download from the University of California at Irvine Machine Learning Repository [52].

The first data set is the Wine data set, which has 178 observations of 13 input dimensions and a single classification dimension, for which there are three possible levels. The second data set is the Iris data set. Perhaps the most widely-used data set in machine learning, the Iris data set consists of 150 observation of four input dimensions, and a single classification dimension, for which there are three possible labels. The input data consists of four measurements of plant physiology, and the classification label is the species of the measured plant. The Iris data set is selected both for its popularity, which provides ample opportunity for comparison to other machine learning techniques, and for the fact that two of the species classifications are not linearly separable in the input dimensionality, while one is not. The classification of the data samples are equally distributed, with 50 samples of each of the three classes, which ensures that any randomly selected subset of the data is likely to also have a equal distribution of classes. Finally, the algorithms performance on the Wisconsin Breast Cancer data set is evaluated. This data set, henceforth called the Cancer data set, is a binary classification problem with 569 observations of a

30-dimensional input vector.

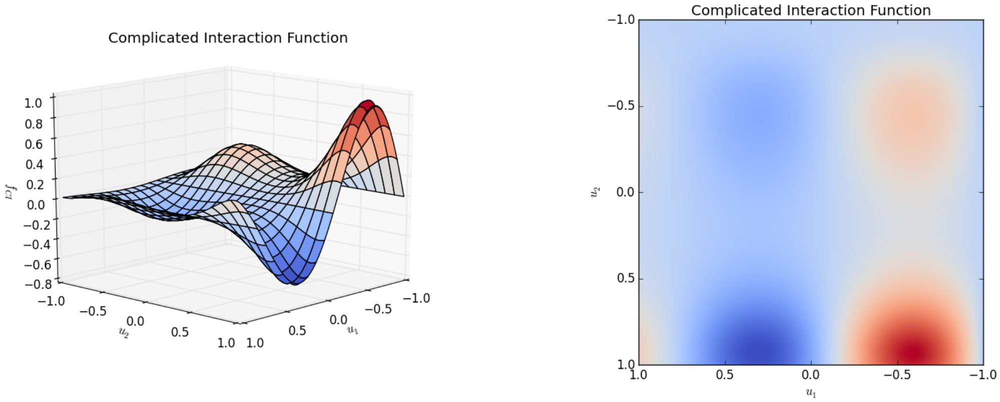
The Wine, Iris, and Cancer data sets are chosen to serve as the trial data sets for classification problems for several reasons. The data sets are examined in other work, and thus enable comparison to previous results. Each problem is relatively small, thereby enabling a robust analysis consisting of a large number of trials, which is essential for evaluating the effectiveness of stochastic procedures. This thesis focuses on the theoretical construction and rigorous evaluation of synaptic annealing, rather than on the application of synaptic annealing ot any particular type of problem. As such, though synaptic annealing is applicable to more complex data sets, this application is deferred to future work.

Regression Data Sets.

Regression performance is evaluated using two two-dimensional functions. The first regression trial function is the complicated interaction function, which is given mathematically by

$$f_{\mathrm{CI}}(u_1, u_2) = 1.9(1.35 + e^{0.5(u_1+1)} \sin(13(0.5u_1 - 0.1)^2) e^{0.5(u_2+1)} \sin(3.5u_2 + 3.5)), \quad (56)$$

and is shown in Fig. 14. This function is chosen for the subtly of the interaction of the two input variable, which creates the potential for high generalization error. The complicated interaction function is also employed in [11], where it is used to characterize the generalization error of a neural network trained using simulated annealing. Thus, using the complicated interaction function also allows for a direct comparison with previously-published results form a methodology which is relatively similar to the one constructed in this thesis. This comparison is of particular interest when evaluating the performance of the weight-anisotropic GSA variant of synaptic annealing, as it is designed to have a similar effect to that of the MOHGSA approach



(Left) **(Right)**

Figure 14. (Left) A plot displaying the surface produced by the complicated interaction function, f_{CI}, in two dimensions (u_1, u_2). (Right) A color contour plot of the complicated interaction function.

advocated in [11] on generalization error. In order to construct a suitable data set for the synaptic annealing algorithm, 100 randomly-selected (u_1, u_2) pairs are drawn uniformly from the range $[-1, 1]$, and the corresponding values of f_{CI} are computed according to Eq. (56).

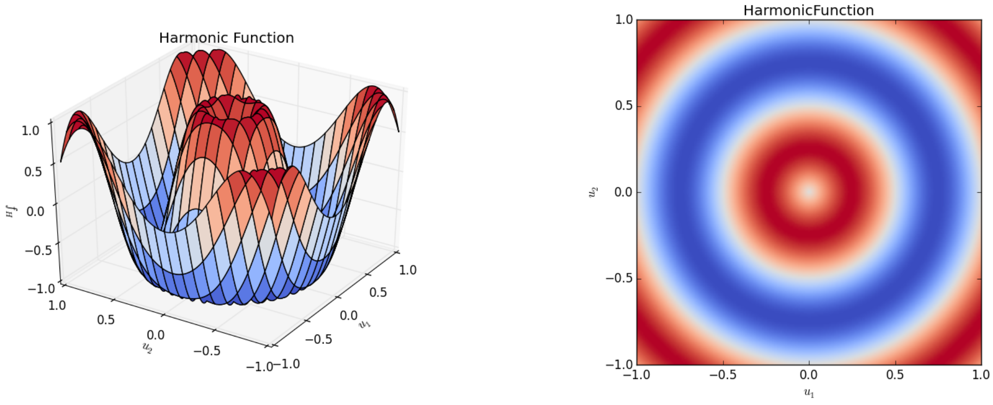
The second regression problem analyzed in this thesis is the harmonic function, which is given by

$$f_H(u_1, u_2) = \sin\left(2\pi\sqrt{u_1^2 + u_2^2}\right), \tag{57}$$

and is shown in Fig. 15. This function was also used in [11], thereby enabling comparison with published results. In [11], the harmonic function is found to be more likely to induce overfitting errors, and consequently yields higher generalization error rates than the complicated interaction function.

Data Preprocessing.

Two preprocessing transforms are applied to the input patterns for all data sets: the mean of the data in that input pattern is removed, and the range of the data



(Left) **(Right)**

Figure 15. (Left) A plot displaying the surface produced by the harmonic function, f_H, in two dimensions (u_1, u_2). (Right) A color contour plot of the harmonic function.

is scaled. For the classification dimension, of which there is only one in each of the data sets used in this thesis, the data labels are orthogonalized. Orthogonalization of classification labels converts a single classification dimension with n possible labels into an n-dimensional representation, in which only one dimension is non-zero for any given data observation. This transform enables regression error-based training algorithms to include false positives and false negatives when computing the error associated with an input pattern. Finally, the order in which observations are presented to the network is random.

Performance Evaluation.

There are several methods available for evaluating the performance of neural networks on a problem set. For classification problems, such as the Wisconsin breast cancer data set, the classification error described in Section 3.4 is used. For regression problems, such as the complex interaction problem, the performance of synaptic annealing is reported in terms of the squared regression error described in Section 53. In this work, a distinction is made between the training cost function, and the reporting cost function. The training cost function is the cost function that is applied to a neural network and data set during the training process in order to inform the training process. The reporting cost function is never used to inform the training process, but will always be used to report the performance of a synaptic configuration on a given data set. This distinction is relevant to the training performed on the classification data sets, for which the regression error can be used to train the algorithm, but is meaningless when trying to determine how effective the resultant neural network is at classifying data set observations.

In this thesis, the performance measure for a synaptic weight configuration, for a given data set, is the n-fold cross validated mean cost function value for that weight configuration and data set. The cost function value is the value of $\mathcal{C}_c$ for classification problems, or $\mathcal{C}_r$ for regression problems. In order to perform a n-fold cross validated evaluation of the algorithm on a given data set the data must be randomly partitioned into n separate sets, often called folds. Of the n folds, one is designated a validation set, and the remaining $(n-1)$ fold are designated the training set. The training set is subset of the data to which the synaptic annealing algorithm has access during training. The validation set is not used by the synaptic annealing algorithm in any way during training. A different FFNN is randomly initialized and trained on each training set. The neural network constructed by the synaptic annealing algorithm is periodically applied to the validation set to produce a time-series of validation set cost function values. These cost function values are an effective analog for the generalization performance of the neural network. To construct a n-fold cross validated estimation of the time series validation set performance, the training process is accomplished n times, each time with a different fold acting as the validation set. The mean and standard deviation, as well as other statistical measures, of these n time series validation set performance measures can then be produced at each time step to reveal how the cost function value a neural network produced by synaptic annealing changes through training time.

Configuration.

The feed forward neural network and simulated annealing algorithms each have a considerable number of tunable parameters which must be specified before they can be applied to a problem. Synaptic annealing, being a synthesis of the two, inherits the parameters of both and, before it can be evaluated, these parameters must be specified. In this section the parameters chosen will be described and the rational for those choices will be given.

Feed Forward Neural Network Configuration.

For all experiments presented in the Section 4.2 and Section 4.2, the trained neural network has an input layer which is the size of the input dimensionality of the data set to which it is being applied, and an output layer which is the size of the number of classes in that data set. The network contains a single hidden layer comprising 20 neurons; the size of the hidden layer was chosen through empirical observation of the performance of the algorithm on the data sets considered in this thesis, using both classification and regression training error. When conducting the experiments to determine the size of the hidden layer, it was observed that the hidden layer size which produced the best performance depended significantly on the cost function used when training the algorithm. For synaptic annealing algorithms trained using the regression error, the size after which larger hidden layers produced no significant gain in performance is about 10; at about 75 hidden layer neurons, the performance begins to degrade significantly. In contrast, synaptic annealing algorithms trained using the classification cost function require about 20 hidden layer neurons to achiever comparable results, though the hidden layer size at which the results begin to degrade remains about the same. Thus, a hidden layer size of 20 neurons is chosen for all classification data sets because it ensures a fair comparison between the best possible performance of synaptic annealing algorithms using both classification and regression training cost functions. The degradation in performance observed, regardless of cost function, once the hidden layer size exceeds approximately 75 neurons is likely a consequence of the size of the search space caused by the inclusion of very large hidden layers. Further investigation is required to determine if it is possible to overcome this limitation and, if so, how. A suggested course of inquiry is detailed in Section 5.2.

Simulated Annealing Configuration.

All variations of SA require the specification of a temperature schedule, which dictates how the scholastic control parameter for the acceptance criterion changes through training time. In this thesis, all results are produced using the FSA temperature schedule given by Eq. (35). The FSA temperature schedule empirically produces the best performance over time scale of evaluation used in this thesis. This improved performance comes at the risk of the CSA algorithm experiencing the freezing problem; this does not, however, emerge as a problem in practice, which may indicate that most local minima on the cost surface of most problems correspond to relatively low-cost configurations. The initial acceptance criterion temperature, $T_0 = 100$, in all experiments. This value is chosen because it is on the order of the largest cost function values encountered during training, for all data sets used in this thesis. This choice of scale for the temperature parameter is essential because is enables the global search of the weight space in the high temperature limit. If the temperature is too small, when compared to the scale of the cost function output values for a given data set, the acceptance criterion would all reject nearly cost-increasing moves, even at the maximum temperature. An initial learning rate $\alpha = 0.01$ is used, and is empirically selected. Additionally, a learning rate schedule is implemented such that the learning rate is instantaneously decreased by a factor of 10 every $100,000$ training epochs. This decrease in learning rate is coupled with a reset of the temperature parameter to its initial value. This process, also known as reannealing [44], is done to ensure that the global search behavior of simulated annealing is replicated each time the learning rate is decreased. Otherwise, a rapid decrease in learning rate could trap the algorithm in a local minimum.

Additional parameter specifications are required for isotropic and anisotropic GSA. Specifically, the dimensionality D and the shape parameter q_V from Eq. (40) must be selected. Here $D = 1$, because the visiting distribution is used to select the change in each synaptic weight independently and, as such, it is the a distribution over a single dimension for any given sampling from the distribution. The shape parameter, q_V, is empirically selected; it is determined that $q_V = 2.6$ produces the lowest-cost distribution in almost all cases; with $q_V = 2.6$, the weight space traversal exhibits almost no local gradient descent. In GSA an additional stochastic control parameter, T_{q_V}, is introduced. As can be seen in Fig. 4 and Fig. 5, this parameter provides additional control over the shape of the visiting distribution. In some implementations of GSA T_{q_V} decays over time, or is linked to the acceptance criterion temperature, both of which result in a search which becomes more local over time. However, it is not necessary to modify this parameter at all during execution, provided that the initial value results in a distribution which is effective for weight space traversal. In this thesis, $T_{q_V}(t) = 1 \; \forall t$, thus is unmodified during training. Future work extending synaptic annealing should investigate the impact of implementing the many possible decay schedules for T_{q_V}. All parameters relevant to the operation of simulated annealing in this thesis are summarized in Table 3.

Table 3. **This table lists the parameters chosen for the synaptic annealing experiments.**

Synaptic Annealing Parameters		
Parameter Type	Parameter	Value
FFNN	Number of Hidden Layers Hidden Layer Size	1 20
SA	Initial Temperature (T_0) Temperature Decay Function Reannealing Frequency	100 T_0/t (where t is epochs elapsed) $100,000$ epochs
Learn Rate	Initial Learning Rate (α) Learning Rate Step Size Learning Decrease Frequency	0.01 0.1 $100,000$ epochs
GSA	q_V T_{q_V} D	2.6 1 1

4.2 Experiment Results

This section presents the results of the experiments conducted to evaluate the performance of synaptic annealing. The section is divided into two subsections; the first presents the results of synaptic annealing training experiments on several classification data sets, while the second evaluates the algorithms performance when applied to regression data sets.

Classification Results.

This section presents the classification performance of FFNNs trained using synaptic annealing. The four synaptic annealing variants described in Section 3.1 are applied to each of the three classification problems described in Section 4.1, using both of the cost functions given in Section 3.4. The converged 10-fold cross validated classification error (mean and standard deviation) obtained from each experimental configuration is summarized in Table 4 and Table 5. The sum of squared weight is not reported as a performance metric in this chapter. This measure of performance is used in the initial exploration of the weight-space exploration behavior. However, the

Table 4. The mean classification error using the regression training cost function.

Classification Error using the Regression Cost Function			
Data Set	$\mathcal{N}$	μ	σ
Wine	Gaussian	0.0052	0.0166
	Cauchy	0.0	0.0
	GSA	0.0105	0.0333
	GSA - WA	0.0211	0.0368
Iris	Gaussian	0.0750	0.1740
	Cauchy	0.0508	0.0657
	GSA	0.0313	0.0442
	GSA - WA	0.0317	0.0334
Cancer	Gaussian	0.0347	0.0165
	Cauchy	0.0400	0.0276
	GSA	0.0382	0.0234
	GSA - WA	0.0259	0.0186

results presented in this section are evaluated in terms of validation set error. Validation set error serves as an estimate of generalization performance, thereby rendering the sum of squared errors redundant.

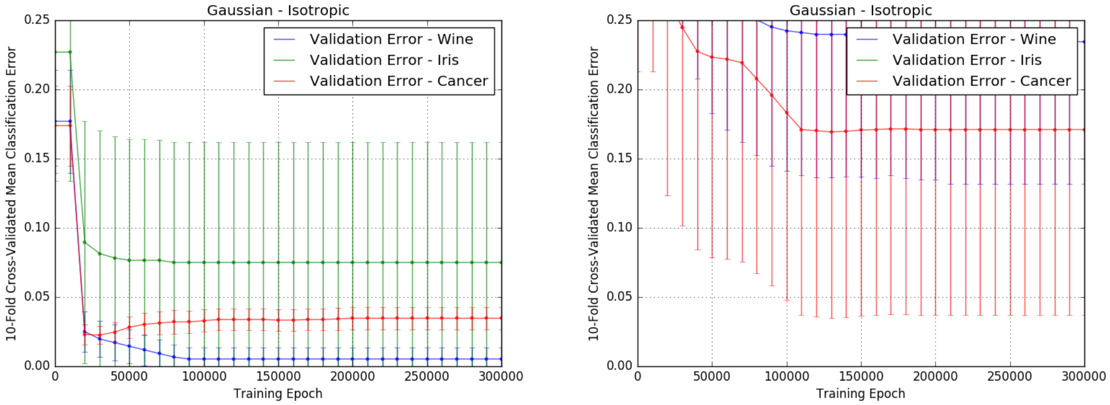
Gaussian Neighborhood Function Results.

In the analysis of simulated annealing variants, the CSA variant of synaptic annealing serves as the baseline for comparison between variants. The CSA variant of synaptic annealing is chosen because it is the variant with the simplest visiting distribution, and is expected to have the worst performance. As such, it is the first variant considered. Fig. 16 displays the time evolution of the 10-fold cross validated validation set classification error for FFNNs trained using synaptic annealing with a Gaussian neighborhood function, using both the classification **(Right)** and SE-based **(Left)** cost functions.

Examination of Fig. 16 reveals several informative classification performance characteristics. Most strikingly, the classification performance achieved by a neural network trained by synaptic annealing employing the classification cost function during

Table 5. The mean classification error using the classification training cost function.

Classification Error using the Classification Cost Function			
Data Set	$\mathcal{N}$	μ	σ
Wine	Gaussian	0.2350	0.2050
	Cauchy	0.0	0.0
	GSA	0.0125	0.0395
	GSA - WA	0.0	0.0
Iris	Gaussian	0.4270	0.2050
	Cauchy	0.0438	0.0422
	GSA	0.0438	0.0593
	GSA - WA	0.0625	0.0977
Cancer	Gaussian	0.1710	0.2680
	Cauchy	0.0552	0.0421
	GSA	0.0362	0.0287
	GSA - WA	0.0362	0.0263



(Left) **(Right)**

Figure 16. Classification error vs. training epoch, where $\mu + \sigma$ are reported for each of the three classification problems (Wine-blue, Iris-green, and Cancer-red), for synaptic annealing using a Gaussian neighborhood function with a (Left) regression and (Right) classification training error function.

training is considerably poorer than one trained using the regression cost function. This phenomenon is produced by the confluence of two factors: the highly-local traversal behavior caused by the Gaussian neighborhood functions and the flat, discontinuous cost surface produced by the classification cost function[1]. Given this explanation,

[1] The classification cost function produces a flat cost surface which changes discontinuously in weight space, because the classification cost function value for a given sample can be only zero or

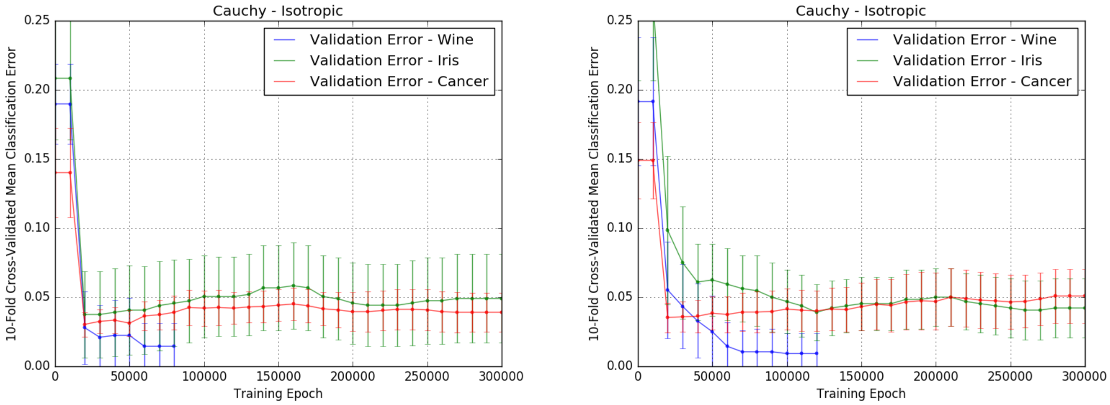
it is anticipated that pseudo-local search algorithms, such as FSA and GSA, should overcome the limitations of the classification training cost function. A A baseline performance for both the classification data sets are established in Fig. 16 (Left). The Wine classification error converges to about 2%, the Cancer classification error converges to approximately 4% and the Iris classification error converges to approximately 7%, with a relatively large standard deviation. The large standard deviation seen with the Iris data set arises from the fact that, of the 10 underling cross-validation training experiments, several converge to perfect, or near-perfect classification, and several become bound in local minima on the cost surface thereby remaining at relatively high classification error values. Again, it is predicted that pseudo-local search algorithms enable escape from these local minima. As such, they should result in classification performance results with considerably lower standard deviations.

Cauchy Neighborhood Function Results.

A neural network trained using synaptic annealing with a Cauchy neighborhood function is expected to exhibit faster convergence to a lower error state, relative to one trained with a Gaussian neighborhood function. This behavior is predicted because of the ability of synaptic annealing using a Cauchy neighborhood function to perform pseudo-local searches in the weight space. In Fig. 17, this prediction is validated. The Cauchy distribution produces neural networks which are able to perfect classify both the training and validation sets for Wine data set in about $100,000$ epochs. Additionally, the mean classification error on the Iris data set is reduced considerably, as well as the standard deviation.

The most substantial difference between the performance of the Gaussian and

one; the observation is either correctly classified, or it is not. Thus, the value of the cost function when applied to a data set, which is the sum of cost function values for each sample, may only take the value of the integers between zero and the cardinality of the data set. Since the cost surface may only take integer values, but is defined on the real-valued range of the synaptic weights, it must be discontinuous and have no gradient between discontinuities.



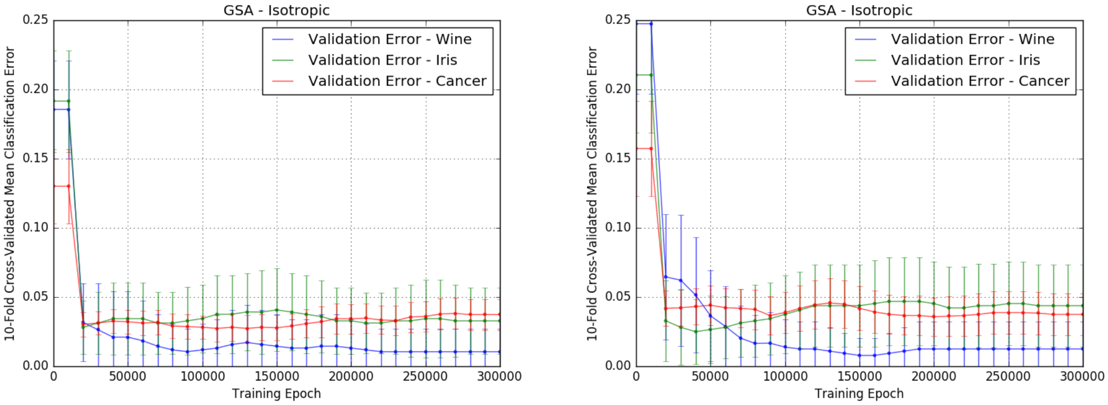
(Left) **(Right)**

Figure 17. Classification error vs. training epoch, where $\mu + \sigma$ are reported for each of the three classification problems (Wine-blue, Iris-green, and Cancer-red), for synaptic annealing using a Cauchy neighborhood function with a (Left) regression and (Right) classification training error function.

Cauchy neighborhood variants of synaptic annealing is the performance of the algorithm when using the classification cost function as the training cost function. The differences in the classification error time evolution between the regression and classification training cost functions seen in Fig. 16 are not present in Fig. 17. This ability to learn from the data using the classification cost function can be attributed directly to the pseudo-local search behavior of Cauchy neighborhood function synaptic annealing, which is in turn caused by the larger tails of the Cauchy distribution.

GSA Neighborhood Function Results.

As discussed in Section 3.6 the GSA neighborhood function results in a more homogeneous search of the weight space, relative to the Cauchy and Gaussian neighborhood functions. Thus, the weight configurations found should, given a sufficient number of epochs, be superior to those found by other neighborhood functions. The results displayed in Fig. 18, Table 5, and Table 4 suggest that this is the case. For both the Iris and Cancer data sets, using both the classification and regression cost function, the GSA neighborhood yields a lower final cross-validated classification error mean and standard deviation.

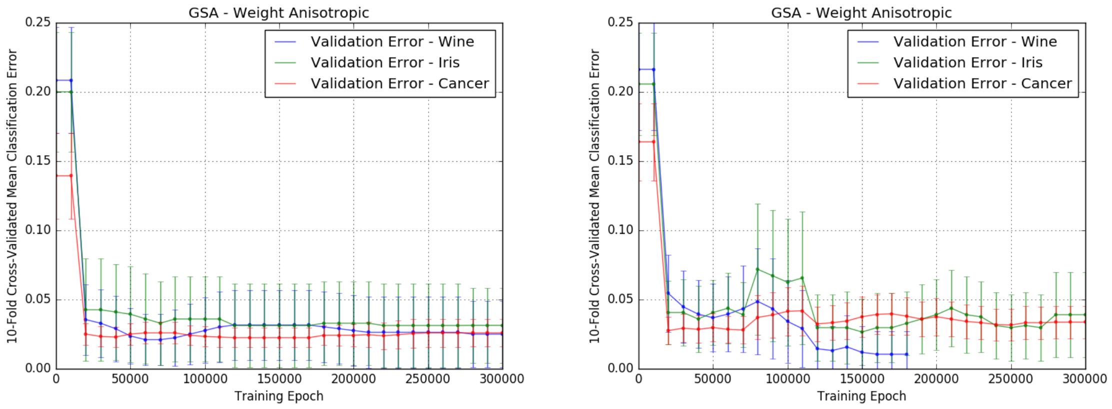


(Left) **(Right)**

Figure 18. Classification error vs. training epoch, where $\mu + \sigma$ are reported for each of the three classification problems (Wine-blue, Iris-green, and Cancer-red), for synaptic annealing using a GSA neighborhood function with a (Left) regression and (Right) classification training error function.

Anisotropic GSA Neighborhood Function Results.

The most notable result attributed to the anisotropic neighborhood function is the mean cross-validated error on the Cancer data set, given a regression cost function. While all other methods fail to achieve a mean error of less than 0.035 on the Cancer data set, the anisotropic GSA neighborhood function achieves a mean error of 0.026 at epoch $300,000$. The Cancer data set is the most challenging used in this analysis, precisely because the number of positive examples is small, compared to the total number of examples. Thus, learning algorithms are likely to overfit on samples belonging to the larger class, thereby increasing the likelihood of poor generalization. The weight-anisotropicity of the neighborhood function ensures that overfitting, while not impossible, is much less likely because all weight values are constrained to remain small, unless making them larger significantly decreases the error.



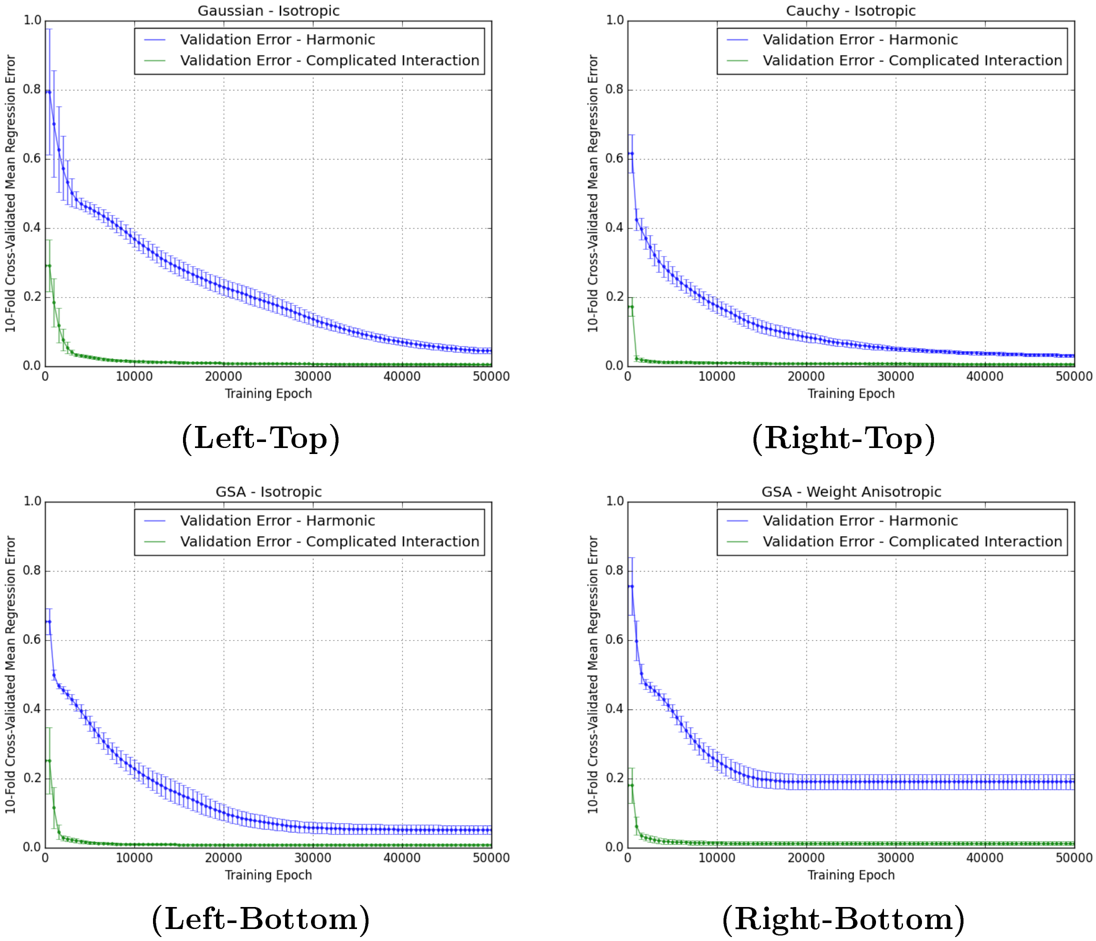
(Left) **(Right)**

Figure 19. Classification error vs. training epoch, where $\mu + \sigma$ are reported for each of the three classification problems (Wine-blue, Iris-green, and Cancer-red), for synaptic annealing using an anisotropic GSA neighborhood function with a (Left) regression and (Right) classification training error function.

Regression Results.

In this section, the results produced by FFNNs trained using various configurations of the synaptic annealing algorithm, when applied to the regression data sets, are evaluated. The complicated interaction and harmonic functions described in Section 4.1 are used to evaluate the performance of each variant of synaptic annealing. The data sets are constructed by randomly and uniformly drawing 500 points from the two-dimensional range of the functions. The regression performance is evaluated using 10-fold cross validation, thus a training set of 400 samples and a validation set of 100 samples are constructed for each fold.

Fig. 20 displays the regression performance of neural networks trained using synaptic annealing. Even the simplest variation of synaptic annealing, shown in Fig. 20 **(Left-Top)**, is able to construct a neural network which closely approximates the trial functions. For all synaptic annealing variants, the complicated interaction function is more easily approximated than the harmonic function. This is likely due to the fact that the harmonic function has considerably more variance than the complicated



(Left-Top)

(Right-Top)

(Left-Bottom)

(Right-Bottom)

Figure 20. This figure contains one plot for each variant of the synaptic annealing training algorithm. Each plot displays the time evolution of the 10-fold cross validated mean and standard deviation of the regression error of neural networks trained using a single synaptic annealing variant on each of the the function approximation data sets. The synaptic annealing variants used in each plot are (Left-Top) Isotropic Gaussian, (Right-Top) Isotropic Cauchy, (Left-Bottom) Isotropic GSA, and (Right-Bottom) Anisotropic GSA.

interaction function.

The differences in regression performance are generally limited to the speed of convergence to the final solution. Fig. 20 reveals that a GSA neighborhood function yields a faster convergence on the harmonic function data set than a Cauchy neighborhood function, which is in turn faster than a Gaussian neighborhood function. The fastest convergence on the harmonic function is achieved by the anisotropic GSA variant of synaptic annealing, though this variant converges to a higher-error solution.

Table 6. This lists of parameter values chosen for the synaptic annealing experiments.

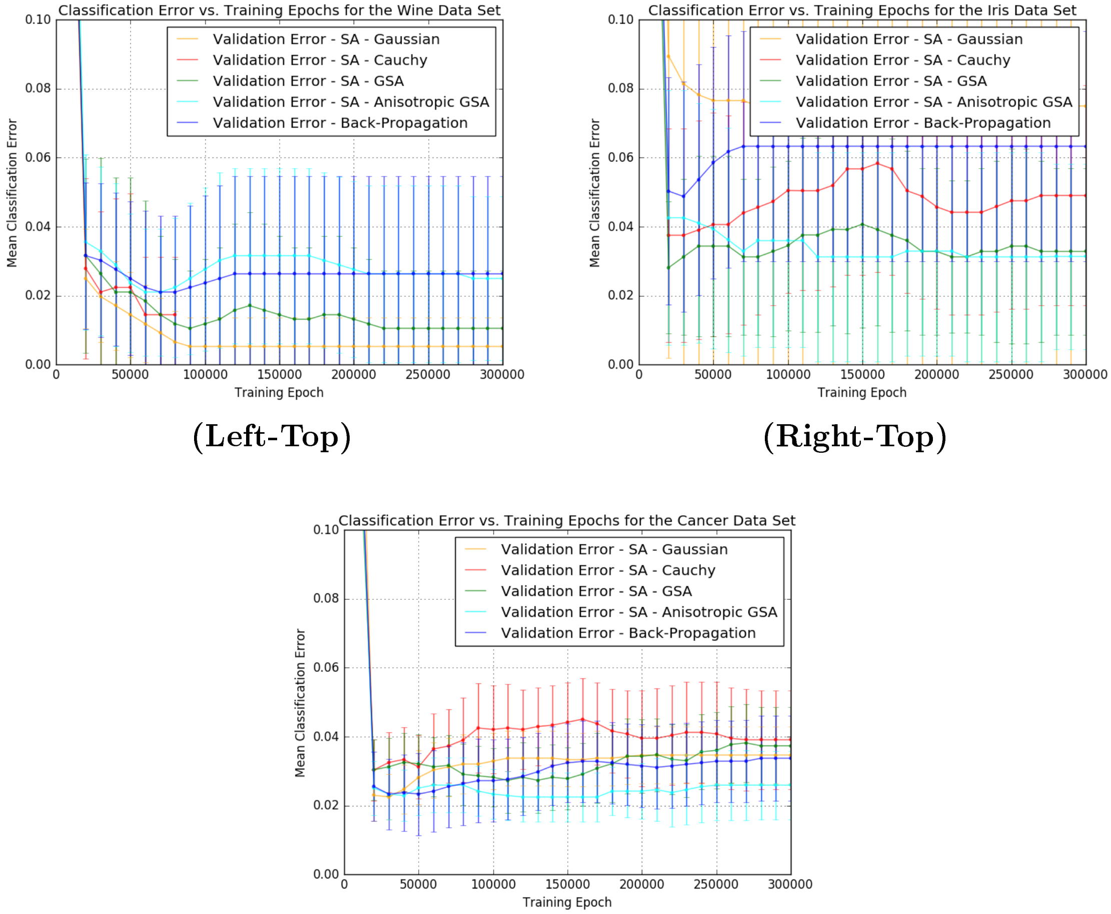
Back-Propagation Parameters	
Parameter	Value
Number of Hidden Layers	1
Hidden Layer Size	20
Initial Learning Rate (α)	0.01
Learning Rate Step Size	0.1
Learning Decrease Frequency	$50,000$ epochs
Momentum Value	0.9

4.3 Back-Propagation Comparison

In order to evaluate the relative effectiveness of synaptic annealing, the results obtained using synaptic annealing are compared to those obtained on the same problems using back-propagation. A neural network is trained using the back-propagation algorithm on each of the classification problems used in this thesis. The back-propagation parameters used in this comparison are listed in Table 6.

The same 10-fold cross validation procedure described in Section 4.1 is used to evaluate the training of FFNNs on each of the classification problems, using back-propagation. Each of the three plots in Fig. 21 corresponds to one of the classification data sets. In each plot, the results obtained from each variant of synaptic annealing are juxtaposed with the results obtained from back-propagation. These synaptic annealing results are the same results displayed in Fig. 16, Fig. 17, Fig. 18, and Fig. 19, grouped by data set, rather than by variant. Thus, Fig. 21 also enables the direct comparison of synaptic annealing variants. In Section 4.2, synaptic annealing variants trained using the regression cost function as the training cost function are generally found to be superior to those using the classification cost function. Thus, only those results are compared in this section.

The results shown in Fig. 21 display several interesting performance characteristics. First, directly comparing back-propagation (blue) to synaptic annealing reveals



(Left-Top)

(Right-Top)

(Bottom)

Figure 21. The classification error ($\mu + \sigma$) vs. training epochs for several FFNN training algorithms for the (Left-Top) Wine, (Right-Top) Iris, and (Bottom) Cancer classification data sets.

that, for the Iris data set **(Right-Top)**, synaptic annealing generally finds a lower-error solution than back-propagation. All variants, except the Gaussian variant, have a lower mean error than back-propagation. Anisotropic GSA synaptic annealing (cyan) produces the lowest-error of all training algorithms; the mean of anisotropic GSA synaptic annealing error is nearly a full standard deviation lower than that of back-propagation. Isotropic GSA synaptic annealing is observed to perform nearly as well as the anisotropic GSA. The Cauchy synaptic annealing variant also outperforms back-propagation, but results in a somewhat higher error than either of the

GSA variants.

A similar relationship is observed between back-propagation and anisotropic GSA synaptic annealing in Fig. 21**(Bottom)**, which displays the results of each algorithm on the Cancer data set. Again, anisotropic GSA synaptic annealing is found produce a considerably lower error than all other evaluated algorithms. Additionally, anisotropic GSA synaptic annealing is observed to be the only training algorithm which produces a lower error than back-propagation on the Cancer data set.

The results obtained for the Wine data set **(Bottom)** present a different trend in performance than those observed for the Iris and Cancer data sets. For the Wine data set, the lowest-error algorithm is synaptic annealing with a Cauchy neighborhood function, which achieves perfect classification in approximately 80000 epochs. Synaptic annealing with a Gaussian neighborhood function produces the second-lowest error. For the Wine data set all variants of synaptic annealing produce lower errors than back-propagation. It should be noted, however, that the high variance shown in the results for back-propagation are caused by the fact that perfect classifications were found for several of the cross-validation folds very quickly, while low errors were never achieved for other folds. This behavior is indicative of the sensitivity to initial weight configurations of gradient descent methods. The large mean and variance of classification error observed for back-propagation, relative to the results for synaptic annealing, highlights the robustness of stochastic search procedures.

4.4 Summary

In this chapter, a design of experiments is constructed and executed in order to evaluate the effectiveness of anisotropic synaptic annealing as a FFNN weight selection algorithm. Several variants of synaptic annealing are rigorously examined, compared, and contrasted. The results obtained indicate that synaptic annealing, when using the anisotropic variant of GSA proposed in this thesis, is superior to both the FSA and CSA variants for several trial problems. Finally, synaptic annealing is found to select lower-error synaptic weight configurations than the back-propagation algorithm for a fixed amount of computational resources.

V. Conclusion

In this chapter the methodology, results, and analysis presented in this thesis is summarized. Relevant conclusions regarding the efficacy of synaptic annealing as a machine learn algorithm are drawn. A discussion on the possibility of future work related to the algorithm developed in this thesis is presented, and several specific, actionable researcher paths are proposed. Finally, the chapter concludes with a brief summary of the contributions made in this thesis.

5.1 Summary of Methods, Results, and Conclusions

The general aims of this thesis are to explore the application of simulated annealing to the problem of feed-forward neural network (FFNN) weight selection, to construct a formalism to concisely and completely describe the combination of SA and FFNNs, and to rigorously evaluate the performance of the resultant machine learning algorithms on several example problems to determine the validity of the methodology. A detailed exposition of the definition and characteristics of both the SA metaheuristic algorithm and FFNNs are presented. A novel formalism for combining the two concepts is established and an algorithmic representations of the formalism is given. In the most general sense, this metaheuristic algorithm, called synaptic annealing, constructs a function which encodes a relation mapping a set of input vectors to a set of output vectors. The synaptic annealing algorithm is left sufficiently general to allow for many variations of SA to be implemented using Algorithm 2.

Algorithm 2 is then implemented using many variants of SA. The behavior of the resultant algorithms are analyzed using visualizations of traversals of a two-dimensional subspace of the complete weight space. The resulting visualizations provide insight, which enable predictions, regarding the cost function minimization performance of the algorithm. These predictions are then evaluated using cross validation on several classification and regression data sets.

5.2 Future Work

In the course of this work, several opportunities for expansion, extension, or modification of the methodology constructed in this thesis were identified. While pursuing those avenues of inquiry were outside the scope of this work, many of them are promising, and could lead to considerable performance enhancements. As such, those suggestions are reproduced here as summary of possible future work

Methodology Extensions.

The first topics which should be studied in any future work extending the work presented are those topics which are minor variations on the methodology presented in Chapter III. A more thorough study of the effects of the variation of parameters in all GSA neighborhood functions is warranted. In this thesis, the problem was scoped such that all GSA visiting distribution implementations used the same values for the parameters q_V, T_{q_V}, and D. Those values were determined by empirically examining the results produced by various parameter values and selecting those which resulted in the best performance. It seems intuitively reasonable that an implementation which uses a decreasing value for T_{q_V}, as is suggested in much of the literature [45, 46], would produce a weight space traversal that is more local through the training epochs. However, caution must be taken to ensure that deleterious interaction effects between a decreasing learn rate and T_{q_V} do not emerge. Further study is required to determine the impact of these effects.

In one application of GSA [49], it is suggested that lowering q_V through training time considerably decreases the time required for convergence of the GSA algorithm for certain types of problems. The problem present in [49] is superficially similar to the weight selection problem, and so this path could yield performance enhancements, and should be considered in future work.

Hybrid Greedy Search Techniques.

Introducing a local greedy optimization algorithm, such as that described in [11], could improve performance, as it would ensure that any local minima found is exploited fully before a global search is resumed. It would be particularly interesting to compare the results of [11] using MOHGSA, to the synaptic annealing methodology present in this thesis using a GSA visiting distribution and employing a gradient descent algorithm which periodically performs steepest-descent on the local error surface.

Reannealing.

Early analysis, not included in the present document, indicate that reannealing produces a modest improvement in performance for all synaptic annealing algorithms. An exploratory study of oscillatory variation in the temperature and learn rate parameters yielded encouraging preliminary results. Further study may yield additional insights into the benefits of alternating global and local distribution characteristics during a semi-local solution space search.

Novel Anisotropicity Policies.

A review of the relevant literature indicates that the concept of anisotropicity, as it relates to traversal through the synaptic weight space during neural network training, is an unexplored concept. The idea is introduced and formalized in this work, but the scope of investigation prevents a thorough exploration of possible implementations of anisotropicity. It is possible that some anisotropicity policy could reduce the size of the synaptic weight space in such a way that the resultant unreachable regions of the space are always of higher cost than the reachable ones. If constructed, such an anisotropicity could result in training algorithms which spend less time searching high cost solutions, and would thereby speed up convergence. One such an anisotropicity policy, weight-based anisotropicity, is evaluated in this work, but others may be constructed and evaluated.

5.3 Contributions

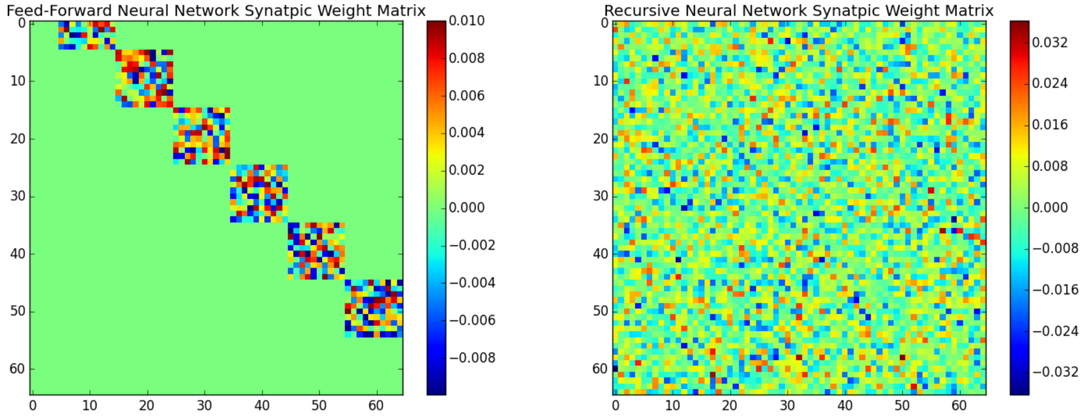
This thesis makes several contributions to the machine learning research community. The first application of GSA to the neural network weight selection is described in this thesis. This application is shown to be effective for a large class of problems, which indicates that it may be fruitfully applied to a broader class of problems. Due to the complexity of generating samples from the GSA distribution, a procedure which quickly produces samples is presented. This procedure is realized as a library, which is applicable to future applications of the GSA distribution. The most complete analysis yet conducted of the application of simulated annealing to the problem of FFNN weight selection is presented in this thesis document. Though other work has explored this combination of algorithms [11], this thesis contains a complete analysis of the algorithm development, as well as a general algorithm (Algorithm 2) which may be implemented and applied to new problems. In the course of developing this algorithm, a formalism of synaptic annealing is constructed, upon which future work can be built.

Appendix A. Recursively-Defined Dot Product Neural Network Propagation

There are many competing conventions [10] for neural network notation and representation. In this appendix, a complete representation for both a neural network data structure, or neural substrate, and the signal propagation operation on that substrate are proposed and described. Two variation of the substrate will be considered, and the the asymptotic computational complexity of the signal propagation algorithm will be analyzed using both substrate representations; the advantages and limitations of each will be explored. The purpose of the notation proposed in this appendix is to concisely and intuitively represent all possible neurocomputational structures, and to explore the computational complexity of algorithms which operate on structures defined using that notation.

1.1 Mathematical Framework

All neural networks are graphs. Thus, it is natural to use the techniques of graph theory to represent and analyze neurocomputational structures. A neural network may be encoded as a weight matrix, wherein each edge weight corresponds to a single synaptic weight in the neural network. This representation preserves all information about the structure of the neural network, and the strengths of the connections between neurons. This synaptic weight matrix serves as the foundation of the neural



(Left-Top) **(Right)**

Figure 22. (Left) A visualization of ω corresponding to a feed-forward neural network. (Right) A visualization of ω corresponding to a recursive neural network.

network notation scheme proposed in this thesis, and will be represented by

$$\omega = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \dots & \omega_{1n} \\ \omega_{21} & \omega_{22} & \omega_{23} & \dots & \omega_{2n} \\ \omega_{31} & \omega_{32} & \omega_{33} & \dots & \omega_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega_{n1} & \omega_{n2} & \omega_{n3} & \dots & \omega_{nn} \end{bmatrix}, \tag{58}$$

where ω_{ij} encodes the synaptic weight of the connection from neuron i to neuron j. Neural networks of different types will produce weight matrices with distinct characteristics. Fig. **??** shows randomly constructed synaptic weight matrices which correspond to feed-forward **(Left-Top)** and recursive **(Right)** neural networks. Note that the synaptic weight matrix corresponding to feed-forward neural network is relatively sparse.

All neural networks operate on data. Thus, a notation is required to represent the data on which the neural networks will operate. Most neural networks are trained by a supervised learning algorithm which seeks to minimize some heuristic cost function.

The cost function is defined on the a pair of data structures: the output produced by a neural network, when presented with an input pattern, and the true output for that input pattern. In the most abstract sense, a well-constructed neural network solves the problem of mapping a set of input patterns to a set of output patterns. An individual instance of the input-output mapping problem can be represented as an ordered pair of input and output patterns, such as

$$(\chi, \lambda) = \{\{\chi\}, \{\chi, \lambda\}\}, \tag{59}$$

where χ is the input pattern, and λ is the true output pattern, or label[1], associated with χ. A neural network training problem is therefore simply a set of ordered pairs, or relation, which will be notated as X in this work. A single pattern-label pair is called an observation, whereas a set of pattern-label pairs is called the data set.

With the synaptic weight matrix and data set notation in place, it is possible to construct a notation which represents the propagation of an input pattern, χ, through the network to produce an output pattern, φ, against which the label pattern, λ, can be compared. First, observe that the propagation through a single neuron is the output the activation function of the neuron, given the sum of the input neurons times the respective synaptic weight for each input, which is given by

$$o = f_A(w_1 i_1 + w_2 i_2 + \cdots + w_m i_m) \tag{60}$$

where o is the output of the neuron, f_A is the activation function, w_n is the synaptic weight value for input n, i_n is the input value for input n, and m is the number of inputs. Clearly, the argument of f_A is the inner produce of two vectors of length m, so it is possible to represent as least the summation component of the propagation of

[1] The symbol λ is adopted in this work because the output pattern in a classification problem is often called the label of the input data.

a single neuron in terms of vectors. Dirac's bra-ket notation for vectors and matrices will be adopted in this work, because of its elegance and conciseness. Thus, a column vector, or ket, which represents the synaptic weight values of the inputs is defined as

$$|w\rangle = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}, \tag{61}$$

and a row vector, or bra, which represents the input values is given by

$$\langle i| = \begin{bmatrix} i_1 & i_1 & \dots & i_m \end{bmatrix}. \tag{62}$$

The argument to f_A is therefore given by

$$\langle i|w\rangle = \begin{bmatrix} i_1 & i_1 & \dots & i_m \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} = (w_1 i_1 + w_2 i_2 + \cdots + w_m i_m). \tag{63}$$

So, the propagation output of an individual neuron can now be expressed as

$$o = f_A(\langle i|w\rangle). \tag{64}$$

An algorithm which accomplishes the operation described by this notation has complexity $O(m)$, because exactly m multiplications, and at most m summations, are required to compute the dot product of two vectors.

This notation can be further extended to include the f_A as a matrix operation, so long as f_A can be approximated by a power series. In order to construct this formalism, a new vector, $|c\rangle$, is introduced. $|c\rangle$ is a vector of length C which contains the first C coefficient of a power series expansion which approximates f_A. Additionally, a column vector containing the sequential powers of the weighted-input dot produce is defined as $|\rho\rangle$ such that

$$|\varrho\rangle = \begin{bmatrix} \langle i|w\rangle^0 \\ \langle i|w\rangle^1 \\ \vdots \\ \langle i|w\rangle^C \end{bmatrix}. \tag{65}$$

Using these vectors, the power series approximation of $f_A(\langle i|w\rangle)$ is given by

$$f_A(\langle i|w\rangle) \approx \langle c|\varrho\rangle = \langle 1| \left(\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_C \end{bmatrix} \circ \begin{bmatrix} \langle i|w\rangle^0 \\ \langle i|w\rangle^1 \\ \vdots \\ \langle i|w\rangle^C \end{bmatrix} \right) = \sum_{j=0}^{C} c_j \langle i|w\rangle^j. \tag{66}$$

The Hadamard product notation is included in Eq. (66) because it will be useful when scaling this framework to describe the propagation of many neurons simultaneously.

In practice, this notation can be realized by an algorithm which runs in $O(C)$ time, because only C multiplications and additions are required. This is slower than the fastest implementation of many activation functions, which run in constant time. However, the notation is convenient, and has two advantages over a standard functional calculation. First, this matrix formulation of the activation function enables each neuron in the network to have a different activation function. Second, the matrix implementation of f_A enables the construction of arbitrary activation functions, effectively enabling the exposure of the activation function as a learned parameter in a metaheuristic weight selection algorithm, such as simulated annealing.

To extend the propagation notation defined in the preceding paragraph to the entire network, recall that the dot product of a row vector and a matrix yields a row vector in which each element is the dot product of the vector and a column of the matrix. Further, observe that in the weight matrix ω, given by Eq. (58), column n represents the synaptic weights of the input synapses neuron n, and is thus analogous to the vector $|w\rangle$ in Eq. (61). Finally, note that X may be represented as $(\langle\chi|, \langle\lambda|)$, where both $\langle\chi|$ and $\langle\lambda|$ are row vectors which have been padded with 0 to be of length n. Thus, $\langle\chi|$ is a vector which represents the input to each neuron in the network, and is therefore analogous to $\langle i|$ in Eq. (62). Combining these observations, it is clear that $\langle\chi|\omega$ yields a row vector in which each element is the weighted sum of inputs to a neuron in the network.

Each element of the weighted sum of inputs vector must be transformed by the activation function of the neuron to which is corresponds. The activation functions of all the neurons in the network can be represented by a single $C \times n$ matrix, c, such as

$$c = \begin{bmatrix} c_{01} & c_{02} & \cdots & c_{0n} \\ c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{C1} & c_{C2} & \cdots & c_{Cn} \end{bmatrix} \tag{67}$$

wherein each column corresponds to the power series coefficients of a single neuron. A complementary $C \times n$ matrix, comprising the sequential powers of the sum of weighted inputs to each neuron, can then be constructed as

$$\rho = \begin{bmatrix} [\langle i|\omega]^{0} \\ [\langle i|\omega]^{1} \\ \vdots \\ [\langle i|\omega]^{C} \end{bmatrix}. \tag{68}$$

Given these matrix definitions, the power series transform of $\langle i|\omega$, and therefore the output of the each neuron in the network, given the input vector i, is given by

$$\langle\varphi| = \langle 1|(c \circ \rho) = \langle 1| \begin{bmatrix} c_{01} & c_{02} & \dots & c_{0n} \\ c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{C1} & c_{C2} & \dots & c_{Cn} \end{bmatrix} \circ \begin{bmatrix} [\langle i|\omega]^{\circ 0} \\ [\langle i|\omega]^{\circ 1} \\ [\langle i|\omega]^{\circ 2} \\ \vdots \\ [\langle i|\omega]^{\circ C} \end{bmatrix}. \tag{69}$$

Thus, $\langle\varphi|$, is a vector in which each element is the output of a neuron in the network. $\langle\varphi|$ can be computed from ω, i, and c in $O(n^2 + Cn)$ time. The two terms of the complexity function arise from the fact that C is arbitrary and therefore may be larger than n. If C is larger than n, the Hadamard product of ρ and c, which runs in $O(Cn)$, would become the fundamental operation, rather than the calculation of $\langle i|\omega$, which requires $O(n^2)$ time. If a fixed activation function which is $O(n^2)$ is used, the complexity of the calculation of $\langle\varphi|$ is $O(n^2)$.

The calculation of $\langle\varphi|$ yields the output of each neuron after an input pattern[2] is propagated into the network. In this framework, each neuron performs a single neurocalcalution, which is a summation of inputs and nonlinear transform of the

[2] In practice, unless the network is a dense graph, most of the elements of the input pattern will be zero. Consider the first propagation of a feed-forward neural network: Only the first layer of neurons will be nonzero. In the second propagation, only the second layer will be nonzero, etc.

sum, during each calculation of $\langle\varphi|$. However, it is generally the case that several sequential propagations are required to completely propagate a pattern through a network. Fortunately, this notation readily incorporates sequential propagations.

To see how sequential propagations can be computed, observe that the structure of the synaptic weight matrix, ω, is such that the output from each neuron, $\langle\varphi|$, is intrinsically formatted to serve as the input pattern pattern for another propagation. Since $\langle\varphi|$ is properly formatted to represent both input and output patterns, a recursive definition of $\langle\varphi|$ can be constructed as

$$\langle\varphi_i| = \langle 1|(c \circ \rho_{i-1}) = \langle 1| \begin{bmatrix} c_{01} & c_{02} & \cdots & c_{0n} \\ c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{C1} & c_{C2} & \cdots & c_{Cn} \end{bmatrix} \circ \begin{bmatrix} [\langle\varphi_{i-1}|\omega]^{\circ 0} \\ [\langle\varphi_{i-1}|\omega]^{\circ 1} \\ [\langle\varphi_{i-1}|\omega]^{\circ 2} \\ \vdots \\ [\langle\varphi_{i-1}|\omega]^{\circ C} \end{bmatrix}, \tag{70}$$

which indicates that the output pattern of the network is a function of the synaptic weights ω, activation functions c, and the previous output pattern of the network φ_{i-1}. This recursion requires a base case, in order to introduce structured patters into the network; that base case is given by

$$\langle\varphi_0| = \langle\chi|, \tag{71}$$

where χ is an input pattern from a data set, as defined in Eq. 59. Thus, the output of a of each neuron in a network after i sequential propagations is given by $\langle\varphi_i|$. The complexity of computing $\langle\varphi_i|$ is $O(in^2 + iCn)$

The propagation notation can be further generalized to include time-series input patterns by observing that an input pattern is just an additional signal from the preceding propagation. So, the recursion can be redefined as

$$\langle\varphi_i| = \langle 1|(c \circ \rho_{i-1}) = \langle 1| \begin{bmatrix} c_{01} & c_{02} & \dots & c_{0n} \\ c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{C1} & c_{C2} & \dots & c_{Cn} \end{bmatrix} \circ \begin{bmatrix} [(\langle\varphi_{i-1}| + \langle\chi_{i-1}|)\omega]^{\circ 0} \\ [(\langle\varphi_{i-1}| + \langle\chi_{i-1}|)\omega]^{\circ 1} \\ [(\langle\varphi_{i-1}| + \langle\chi_{i-1}|)\omega]^{\circ 2} \\ \vdots \\ [(\langle\varphi_{i-1}| + \langle\chi_{i-1}|)\omega]^{\circ C} \end{bmatrix}, \tag{72}$$

with a base case of

$$\langle\varphi_0| = \langle 0|. \tag{73}$$

This formulation has asymptotically equivalent time complexity to the single-input form.

The final form of the notation is somewhat cumbersome due to the power series expansion computation. In order to reduce this complexity, a new operator, $\textcircled{f}$, which performs the Hadamard power matrix expansion, the Hadamard product, and the unit dot product. This new operator will be called the transformation operator, and is defined as

$$\langle\varphi_i| = \langle 1|(c \circ \rho_{i-1}) \equiv c\textcircled{f}\langle\varphi_{i-1} + \chi_{i-1}|\omega, \tag{74}$$

and should be read as c *transforms* $\langle\varphi_{i-1} + \chi_{i-1}|\omega$. The transformation operator is a binary operator which requires that the left argument is a matrix and the right argument is a row vector of the same width as the right argument matrix.

With the preceding mathematical framework fully constructed, it is now possible to construct an algorithmic representation of recursion defined in equations 70 and 71. Because time series inputs will not be used in the scope of this work, and because they do not alter the asymptotically complexity of the algorithm, all algorithmic analysis will be restricted to equations 70 and 71. A pseudocode representation of these equations is given in Algorithm 3. The pseudocode representation of the `transform` subroutine, which implements the transform operator defined in Eq. (74), is shown in Algorithm 4. The fixed activation function form of the algorithm is recovered by replacing `transform`$(c, \cdot)$ with $f_A(\cdot)$.

Algorithm 3 A pseudocode representation of the recursive dot product neural network pattern propagation algorithm. *Inputs*: i, the number of propagations to be completed, which is also the recursion depth. ω, an $n \times n$ synaptic weight matrix. c, a $C \times n$ matrix encoding the power series coefficients for each neurons activation function. χ, a vector of length n encoding the input pattern. *output*: a vector of length n encoding the output pattern of the neural network defined by ω and c, given i propagations and the input pattern χ.

```
procedure DOTPROP($i$, $\omega$, $c$, $\chi$)                                  ▷
    if $i = 0$ then return $\chi$                        ▷ Return the input pattern.
    else return transform($c$, dot(DOTPROP($i-1, \omega, c, \chi$), $\omega$) )
    end if
end procedure
```

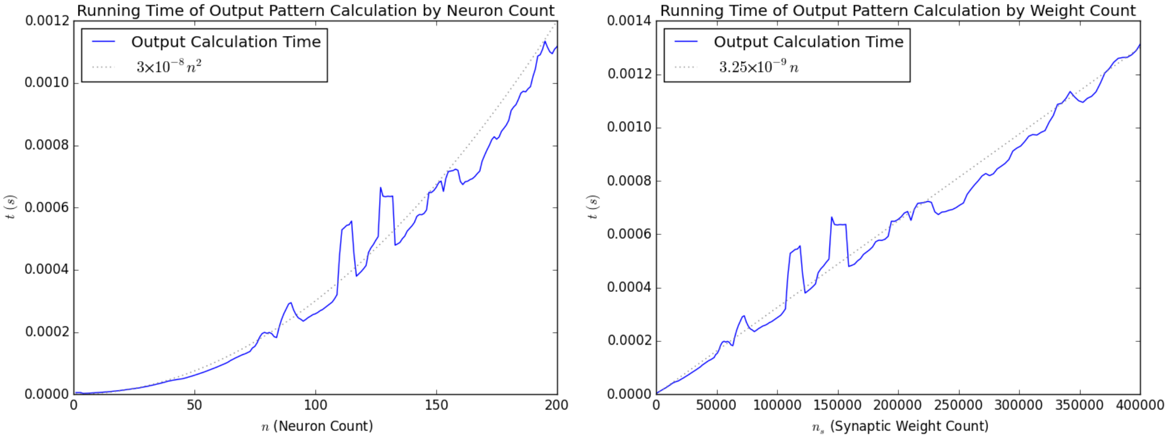
Algorithm 4 A pseudocode representation of the matrix-based power series transformation algorithm. *Inputs*: c, a $C \times n$ matrix encoding the power series coefficients for each neurons activation function. v, a vector of length n encoding the pattern which is transformed by the power series expansion using the coefficients stored in c. *output*: a vector of length n encoding the power-series transformed values of the input vector, v.

```
procedure TRANSFORM($c$, $v$)
    $i \leftarrow -1$                                                   ▷ Initialize $i$
    $\rho \leftarrow [(v^{i+=1})$ for $row \in$ rows($c$)$]$         ▷ Construct the power matrix
    return sum($c \circ \rho$ ,1)    ▷ Return the column sum of the element-wise matrix product
end procedure
```

The theoretically-predicted time complexity of the calculation of $\langle \varphi_i|$ was evaluated experimentally, and the results are show in Fig. 23. The results agree well with the predicted $O(n^2)$ complexity[3]. As can be seen in Fig. 23 (Left), the running time

[3] All results were produced using a system in which $C < n$.

of the algorithm calculating the output pattern, $\langle\varphi_i|$, is $O(n^2)$.



(Left) **(Right)**

Figure 23. (Left) The running time of the Algorithm 3 in n, the number of neurons in the network. (Right) The running time of the Algorithm 3 in n_s, the number of synaptic weights in the network, which, in this network is equal to $n^2 - n$.

1.2 Application to Feed-Forward Neural Networks

This notation can be easily applied to describe the operation of feed-forward neural networks. Consider a feed-forward network with L layers. This results in a weight matrix, ω, which is a block matrix with $L-1$ adjacent blocks, as can be seen in Fig. **??** (Left). Thus, $L-1$ propagations are required to propagate an input pattern through the network, as $L-1$ propagations will result in a dot product applied to each of the blocks. Using this notation, the notation for the output of propagating a pattern through a feed-forward neural network is simply $\langle\varphi_{L-1}|$. The time complexity of feed-forward network propagation is then $O((L-1)(n^2+Cn))$ which reduces to $O(Ln^2+LCn)$.

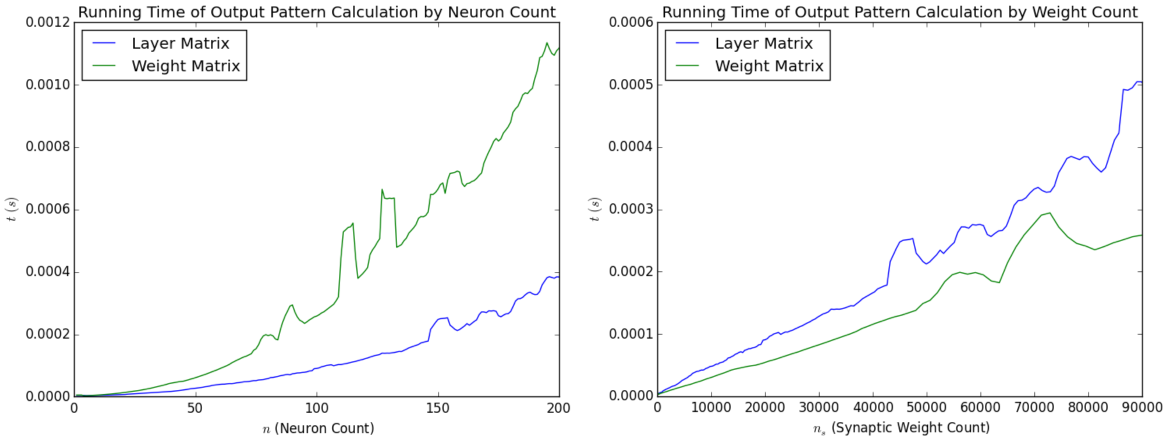
A significant portion of the computation required to compute the propagation through a feed-forward neural network using the preceding framework is unnecessary. Because the synaptic weight matrix of a feed-forward neural network is sparse, due to the lack of recursive connections, most of the multiplications in the dot product computation involve a zero, and therefore do not impact the propagation pattern. The general propagation notation framework can be modified slightly to gain dramatic reductions in computational complexity for some feed-forward neural networks. Consider that the connection between each layer in a feed forward neural network is a block matrix in the upper triangle of the synaptic weight matrix. The propagation and transformation operations only need to operate on these block matrices, and never on the entire synaptic matrix. Thus, in the specific case of a feed forward neural network, these block matrices should be excised from the weight matrix, in blocks that are the size of the largest layer in the neural network. The recursive formula for calculating the output pattern produced by propagating a pattern through these networks is unchanged, except that the weight matrix is a different, equally-sized, block matrix at each recursive depth. This set of block matrices can be easily represented a three dimensional matrix of size $n_{max} \times n_{max} \times (L-1)$, which will be called layer matrices. Also, time series inputs are not generally allowed in feed forward neural networks, so this feature is removed. This recursive relationship is given by

$$\langle \varphi_i | = c \textcircled{f} \langle \varphi_{i-1} | \omega_i, \tag{75}$$

with a base case of

$$\langle \varphi_0 | = \langle \chi_0 |. \tag{76}$$

The layer matrix representation of a feed-forward neural network ensures that the largest matrix considered is of size $n_{max} \times n_{max}$, where n_{max} is the number of neurons in the largest layer. The computational complexity associating with completing a full propagation through the network is therefore $O(Ln_{max}^2 + LCn_{max})$. This predicted computational complexity is validated in Fig. 24 which shows the running time performance of the layer matrix representation of recursive dot propagation algorithm. Fig. 24 also displays the running time performance of the standard dot propagation algorithm; it is clear that both versions of the algorithm are of running time complexity $O(n^2)$, with respect to the number of neurons in the network, as expected. However, the layer matrix version of the algorithm has a considerably smaller multiplicative constant. Interestingly, when considering the time complexity of the two algorithms in the number of synaptic weights, the layer matrix has a higher multiplicative constant. This phenomena emerges from the fact that a feed-forward neural network is necessarily more sparsely connected that a fully-connected recursive neural network, for a given number of neurons.



(Left) **(Right)**

Figure 24. (Left) The running time of the Algorithm 3, employing a layer matrix representation, in n, which is the number of neurons in the network. (Right) The running time of the Algorithm 3, employing a layer matrix representation, in n_s, which is the number of synaptic weights in the network.

Note that n_{max} is guaranteed to be less than n, and that the speedup, s, that results from this change is given by

$$\frac{O(Ln^2 + LCn)}{O(Ln_{max}^2 + LCn_{max})} \propto \frac{n^2}{n_{max}^2}. \tag{77}$$

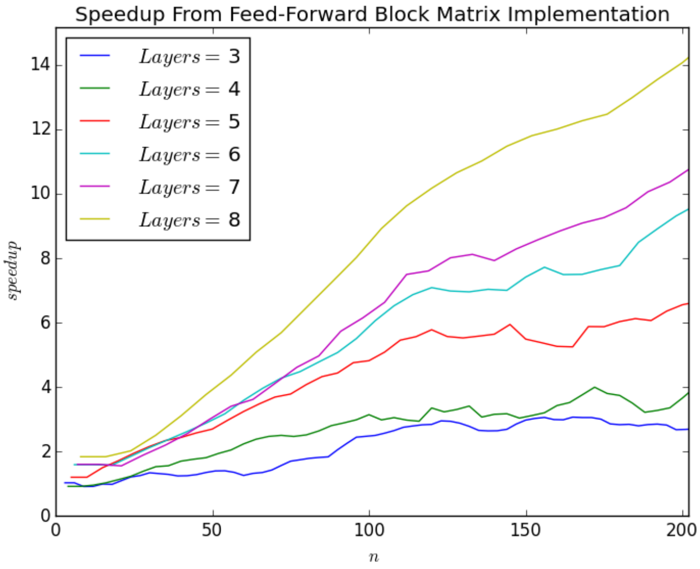


Figure 25. The speedup obtained by employing the layer matrix version of the dot propagation algorithm, relative to the weight matrix version of the algorithm, for a feed-forward neural network. The speedup was calculated for networks of varying depth and size.

Observe, also, that $n \propto L$ because each layer must comprise at least a single neuron; thus as $L \rightarrow \infty$, $s \rightarrow \infty$. Thus, adding a layer increases the computational complexity linearly, while increasing the number of learned parameters in the network by a factor of n_{max}^2. The theoretically predicted computational complexity was experimentally validated; the results of these experiments are shown in Fig. **??**. The networks from which the results in Fig. **??** are derived were constructed by selecting a number of layers and then increasing the value of n_{max} sequentially. Each layer count, L, and n_{max} value, a network with L layers and n_{max} neurons per layer was constructed. The performance of the constructed network was evaluated using both the full-matrix and block-matrix versions of the propagation algorithm, and the speedup was calculated.

1.3 Implementation Considerations

The matrix dot product, which will be the fundamental operation in any implementations of this framework, is readily parallelizable. A thread can be spawned to solve each column, as there is no interdependence between columns. This reduces the computation complexity of calculating $\langle\varphi_i|$ to $O(in + iC)$, assuming that there are more available threads than neurons in the network.

Additionally, many neural networks training algorithms require the calculation of the squared Euclidean distance between a desired output vector, $\langle\lambda|$, and the actual output vector, $\langle\varphi|$. This quantity is easily obtained using the notation developed in this work. Consider that the Euclidean distance from an arbitrary vector, v, to origin is given by $\langle v|v\rangle$, as this quantity is the sum of the squared elements of v, minus 0. Further, observe that the elements of the vector $\langle\lambda - \varphi|$ are simply the difference between the corresponding elements of $\langle\lambda|$ and $\langle\varphi|$. Then, the squared Euclidean distance between $\langle\lambda|$ and $\langle\varphi|$ is given by $\langle\lambda - \varphi|\lambda - \varphi\rangle$. The computation of this quantity is a dot product, and therefore requires only $O(n)$ time, where $n = |\langle\lambda|| = |\langle\varphi||$.

Bibliography

1. P. Lops, M. de Gemmis, and G. Semeraro, “Content-based recommender systems: State of the art and trends.” in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer, 2011, pp. 73–105. [Online]. Available: http://dblp.uni-trier.de/db/reference/rsh/rsh2011.html#LopsGS11

2. M. Richardson, A. Prakash, and E. Brill, “Beyond pagerank: machine learning for static ranking,” in *WWW ’06: Proceedings of the 15th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2006, pp. 707–715. [Online]. Available: http://portal.acm.org/citation.cfm?id=1135777.1135881

3. G. Klambauer, “Machine learning techniques for the analysis of high-throughput dna and rna sequencing data.” Ph.D. dissertation, UniversitÃď't Linz, 2014.

4. T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*. Springer Verlag, August 2001.

5. S. B. Kotsiantis, “Supervised machine learning: A review of classification techniques.” *Informatica (Slovenia)*, vol. 31, no. 3, pp. 249–268, 2007. [Online]. Available: http://dblp.uni-trier.de/db/journals/informaticaSI/informaticaSI31.html#Kotsiantis07

6. X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg, “Top 10 algorithms in data mining,” *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1007/s10115-007-0114-2

7. J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85 – 117, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

8. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

9. T. Weyand, I. Kostrikov, and J. Philbin, “Planet - photo geolocation with convolutional neural networks,” *ArXiv e-prints*, Feb. 2016.

10. S. Haykin, *Neural networks: a comprehensive foundation.* Upper Saddle River, N.J: Prentice Hall, 1999.

11. Y. Lee, J.-S. Lee, S.-Y. Lee, and C. H. Park, “Improving generalization capability of neural networks based on simulated annealing.” in *IEEE Congress on Evolutionary Computation.* IEEE, 2007, pp. 3447–3453. [Online]. Available: http://dblp.uni-trier.de/db/conf/cec/cec2007.htm

12. J. Engel, “Teaching feed-forward neural networks by simulated annealing,” *Complex Syst.*, vol. 2, no. 6, pp. 641–648, Dec. 1988. [Online]. Available: http://dl.acm.org/citation.cfm?id=65512.65514

13. W. S. McCulloch and W. Pitts, “Neurocomputing: Foundations of research,” J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. A Logical Calculus of the Ideas Immanent in Nervous Activity, pp. 15–27. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104377

14. G. Piccinini, “Computational explanation in neuroscience,” *Synthese*, vol. 153, no. 3, pp. 343–353, 2006.

15. N. Rochester, J. Holland, L. Haibt, and W. Duda, “Tests on a cell assembly theory of the action of the brain, using a large digital computer,” *Information Theory, IRE Transactions on*, vol. 2, no. 3, pp. 80–93, September 1956.

16. W. James, *The Principles of Psychology*, ser. American science series: Advanced course. H. Holt, 1890, no. v. 1. [Online]. Available: https://books.google.com/books?id=deOLTt-dD3gC

17. D. O. Hebb, *The organization of behavior; a neuropsychological theory, (by) D.O. Hebb. Science Editions.* New York: John Wiley and Sons, 1967.

18. F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

19. J. A. Anderson and E. Rosenfeld, Eds., *Neurocomputing: Foundations of Research.* Cambridge, MA: MIT Press, 1988. [Online]. Available: http://mitpress.mit.edu/book-home.tcl?isbn=0262510480

20. H. D. Block, “The perceptron: A model for brain functioning,” *Reviews of Modern Physics*, vol. 34, pp. 123–135, 1962.

21. B. Widrow and M. E. Hoff, “Adaptive switching circuits,” in *1960 IRE WESCON Convention Record, Part 4*, Institute of Radio Engineers. New York: Institute of Radio Engineers, 8 1960, pp. 96–104. [Online]. Available: http://www-isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf

22. M. Minsky and S. Papert, *Perceptrons.* Cambridge, MA: MIT Press, 1969.

23. J. A. Anderson, “A simple neural network generating an interactive memory,” *Mathematical Biosciences*, vol. 14, pp. 197–220, 1972.

24. T. Kohonen, “Correlation matrix memories,” *IEEE Trans. Comput.*, vol. 21, no. 4, pp. 353–359, Apr. 1972. [Online]. Available: http://dx.doi.org/10.1109/TC.1972.5008975

25. C. von der Malsburg, “Self-organization of orientation sensitive cells in the striate cortex.”

26. S. E. Brodie, B. W. Knight, and F. Ratliff, “The response of the Limulus retina to moving stimuli: a prediction by Fourier synthesis,” *J Gen Physiol*, vol. 72, no. 2, pp. 129–66, Aug. 1978.

27. J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, 1982. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/6953413]

28. D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cognitive Science*, vol. 9, no. 1, pp. 147–169, 1985.

29. A. G. Barto, R. S. Sutton, and C. W. Anderson, “Artificial neural networks,” J. Diederich, Ed. Piscataway, NJ, USA: IEEE Press, 1990, ch. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, pp. 81–93. [Online]. Available: http://dl.acm.org/citation.cfm?id=104134.104143

30. P. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” Ph.D. dissertation, Harvard University, 1974.

31. A. E. Bryson and Y. C. Ho, *Applied Optimal Control.* New York: Blaisdell, 1969.

32. D. B. Parker, “Learning logic,” Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. TR-47, 1985.

33. Y. Lecun, “Une procédure d’apprentissage pour réseau à seuil asymétrique,” *Proceedings of Cognitiva 85, Paris*, pp. 599–604, 1985.

34. D. Rumelhart, G. Hinton, and R. Williams, *Learning Internal Representations by Error Propagation*, 1986.

35. G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02551274

36. R. S. Sexton, R. E. Dorsey, and J. D. Johnson, “Beyond back propagation: Using simulated annealing for training neural networks,” *J. End User Comput.*, vol. 11, no. 3, pp. 3–10, Jul. 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=329748.329752

37. S. S. Beheraa and S. Chattopadhyay, “A comparative study of back propagation and simulated annealing algorithms for neural net classifier optimization,” *Procedia Engineering*, vol. 28, pp. 448–455, 2012.

38. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

39. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of State Calculations by Fast Computing Machines,” vol. 21, pp. 1087–1092, Jun. 1953.

40. W. L. Goffe, G. D. Ferrier, and J. Rogers, “Global optimization of statistical functions with simulated annealing,” *Journal of Econometrics*, vol. 60, pp. 65–99, 1994.

41. A. Corana, M. Marchesi, C. Martini, and S. Ridella, “Minimizing multimodal functions of continuous variables with the &ldquo;simulated annealing&rdquo; algorithmcorrigenda for this article is available here,” *ACM Trans. Math. Softw.*, vol. 13, no. 3, pp. 262–280, Sep. 1987. [Online]. Available: http://doi.acm.org/10.1145/29380.29864

42. A. Lecchini-Visintini, J. Lygeros, and J. Maciejowski, “Simulated Annealing: Rigorous finite-time guarantees for optimization on continuous domains,” *ArXiv e-prints*, Sep. 2007.

43. H. Szu and R. Hartley, “Fast simulated annealing,” *Physics Letters A*, vol. 122, no. 3-4, pp. 157–162, Jun. 1987. [Online]. Available: http://dx.doi.org/10.1016/0375-9601(87)90796-1

44. L. Ingber, “Very fast simulated re-annealing,” 1989.

45. C. Tsallis and D. A. Stariolo, “Generalized simulated annealing,” *Physica A: Statistical and Theoretical Physics*, vol. 233, no. 1-2, pp. 395–406, Nov. 1996. [Online]. Available: http://www.sciencedirect.com/science/article/B6TVG-3YK5TC8-2H/1/d040f1408073d6a09dc185f38673e3dd

46. A. Dall’Igna J, R. S. Silva, K. C. Mundim, and L. E. Dardenne, “Performance and parameterization of the algorithm Simplified Generalized Simulated Annealing,” *Genetics and Molecular Biology*, vol. 27, pp. 616 – 622, 00 2004.

47. J. A. Freitez, M. Sanchez, and F. Ruette, “Comparative analysis of simulated annealing (sa) and simplified generalized sa (sgsa) for estimation optimal of para-

metric functional in cativic," *AIP Conference Proceedings*, vol. 1148, no. 1, pp. 404–407, 2009.

48. P. D. Moral and L. Miclo, "On the convergence and applications of generalized simulated annealing," *SIAM Journal on Control and Optimization*, vol. 37, no. 4, pp. 1222–1250, 1999.

49. Y. Xiang, D. Sun, W. Fan, and X. Gong, "Generalized simulated annealing algorithm and its application to the thomson model," *Physics Letters A*, vol. 233, no. 3, pp. 216 – 220, 1997.

50. S. Mukherjee and B. K. Chakrabarti, "Multivariable optimization: Quantum annealing and computation," *European Physical Journal Special Topics*, vol. 224, p. 17, Feb. 2015.

51. I. Andricioaei and J. E. Straub, "Generalized simulated annealing algorithms using tsallis statistics: Application to conformational optimization of a tetrapeptide," *Phys. Rev. E*, vol. 53, pp. R3055–R3058, Apr 1996. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.53.R3055

52. C. L. Blake and C. J. Merz, "UCI repository of machine learning databases," http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From — To)*
6–17–2016	Master's Thesis	Jan 2015 — June 2016

4. TITLE AND SUBTITLE

Synaptic Annealing: Anisotropic Simulated Annealing and its Application to Neural Network Synaptic Weight Selection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Fletcher, Justin R., 1st Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-16-J-060

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Neural networks are one of the most successful classes of machine learning algorithm, and have been applied to solve problems previously considered to be the exclusive domain of human intellect. Several methods for constructing neural networks exists. This research explores the effectiveness of a feed-forward neural network weight selection procedure called synaptic annealing. Synaptic annealing is the application of the simulated annealing algorithm to the problem of selecting synaptic weights. A novel extension of the simulated annealing algorithm, called anisotropicity, is defined and developed. The cross-validated performance of each synaptic annealing algorithm is evaluated, and compared to back-propagation when trained on several typical machine learning problems. Synaptic annealing is found to be more effective than back-propagation training on classification and regression data sets. These improvements in feed-forward neural network training performance indicate that synaptic annealing may be a viable alternative to back-propagation in many applications of neural networks.

15. SUBJECT TERMS

Neural Networks, Simulated Annealing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Gilbert L. Peterson, AFIT/ENG
U	U	U	U	127	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-3636 x4281; gilbert.peterson@afit.edu