Scholars' Mine

Doctoral Dissertations

Student Theses and Dissertations

Fall 2011

Spiking neural networks and their applications

Cameron Eric Johnson

Follow this and additional works at: https://scholarsmine.mst.edu/doctoral_dissertations

Part of the Computer Engineering Commons

Department: Electrical and Computer Engineering

Recommended Citation

Johnson, Cameron Eric, "Spiking neural networks and their applications" (2011). *Doctoral Dissertations*. 78.
https://scholarsmine.mst.edu/doctoral_dissertations/78

This thesis is brought to you by Scholars' Mine, a service of the Missouri S&T Library and Learning Resources. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

SPIKING NEURAL NETWORKS AND THEIR APPLICATIONS

by

CAMERON ERIC JOHNSON

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

2011

Approved
Ganesh Kumar Venayagamoorthy, Advisor
Daryl Beetner
Keith Corzine
Thomas Vojta
Donald Wunsch

ABSTRACT

Artificial neural networks (ANNs) have been developed as adaptable, robust function approximators for at least the last quarter-century. They have progressed through two generations, and the third is now under development. Spiking neural networks (SNNs) seek to improve on previous generations in two ways: by using a more biologically-inspired neuron, they are shown to be capable of more complex calculations; incorporating polychronous properties of highly-recurrent networks with delays of different lengths on each synapse to achieve large numbers of possible patterns with relatively few neurons and synapses.

Abstracted spiking neurons have been used as a third-generation activation function in a traditional feedforward network architecture, and their potency in application to a real-world problem – identification of power system generator dynamics – is demonstrated in this dissertation in comparison to a standard sigmoidal multi-layer perceptron network. However, the goal of SNNs is to be able to utilize biological-like neural network modeling to capture the computational prowess of living brains. In order to achieve such a feat, first a bio-inspired SNN must be able to handle continuous-valued function approximation; until this is done, such networks cannot even be compared to their second-generation predecessors.

This dissertation demonstrates a technique for using a faithfully modeled SNN on continuous-valued problems. The encoding and decoding frameworks developed in this dissertation for the biologically-inspired SNN enables it, like any other ANN, to be applied to any time-dependent problem, including neuroidentification of power system generator dynamics.

ACKNOWLEDGMENTS

Special thanks to Dr. G. K. Venayagamoorthy for serving as primary advisor on this Ph.D. research, for encouraging the author to attend multiple conferences and expand his understanding of the field and how research is done.

Additionally, thanks to each member of the advisory committee: Drs. D. Wunsch, D. Beetner, K. Corzine, and T. Vojta for their review of this work and the advice given over the course of this research.

Thanks, as well, to the Real-Time Power and Intelligent Systems Laboratory and the Intelligent Systems Center at Missouri S&T, GAANN grant #P200A070504, and the National Science Foundation Emerging Frontiers in Research and Innovation #0836017 for funding and support throughout the author's Ph.D. candidacy career.
To Sinchan Roychowdhury for his assistance in testing the encoding and decoding methods, and general assistance with coding and developing the experiments and case studies presented here, and to Bipul Luitel for his assistance in simulating power system data, thanks are additionally extended.

To Dr. S. Potter and his students R. Ortman and R. Zeller-Townsend at the Georgia Institute of Technology's Neuroengineering lab, for their help in understanding how living neural networks function, the author extends his thanks.

Finally, to the author's parents and siblings and again to his advisor, he wishes to extend thanks for patience with his discouragement at various points, and for their encouragement and confidence that he could finish this work.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

NOMENCLATURE

Symbol	Description
ADALINE	Adaptive Linear Neuron
GPU	Graphics Processing Unit
ANN	Artificial Neural Network
MLP	Multi-Layer Perceptron
RBF	Radial Basis Function network
RNN	Recurrent Neural Network
TDNN	Time-Delay Neural Network
SNN	Spiking Neural Network
ASNN	Abstracted Spiking Neural Network
BSNN	Biological-model-based Spiking Neural Network
PSN	Polychronous Spiking Network

1. INTRODUCTION

1.1. INTRODUCTION

Neural networks have been studied since man first appreciated that the brain is, itself, a computational engine. The very earliest efforts at developing an integrate-and-fire model of a neuron were introduced in 1907 [1]. The McCulloch-Pitts model was developed in 1943, and a giant squid neuron was dissected to develop the Hodgkins-Huxley model in 1952 [2]. Studying the mechanism of a biological neuron and the structure of living brains (biological neural networks) led to the development of artificial neural networks (ANNs).

1.1.1. Generations of Neural Networks.

ADALINE, developed in 1960, was the first function approximating network of artificial neurons. Using the McCulloch-Pitts model, it had a binary thresholding function. Capable of any sort of binary-encoded function approximation, it is robust and adaptable. This marked the first generation of ANNs [3]. The second generation came about through two breakthroughs in the 70s, with Werbos's work on backpropagation through time [4], [5] and the development of sigmoidal threshold functions. ADALINE and the McCulloch-Pitts neuron use a simple thresholding function that returns a binary value; the second-generation ANN utilized neurons with activation functions capable of outputting continuous values [6].

Advances in neural architectures introduce recurrence and feedback structures, where the outputs of prior time steps (or even the same time step!) help influence the current output. These advances introduce a form of memory to the neural network. The third generation of ANN is currently still being developed: spiking neural networks (SNNs) [3].

Research into SNNs has boomed in the last decade or so, becoming one of the hottest topics at neural networks conferences. Development of new mathematical models for the spiking neuron, investigation and experimentation with various architectures for SNNs, exploration of new platforms such as graphics processing units (GPUs) and other high-performance computing (HPC) clusters [7], and even novel applications for the strongest area for state-of-the-art SNNs: pattern recognition and classification [8], [9]. These categorization problems do not require continuous-valued functional mapping, and include logic gates, [10], [11], [12], video and image processing, [13], [14], [15], [16], [17] or even auditory [18] or gustatory [19] discernment.

1.1.2. Applications. The three most common applications for neural networks are function approximation, pattern recognition, and nonlinear control. Time series prediction and system identification are subsidiary applications to these. As universal function approximators, ANNs' chief advantage is in their adaptability.

Time series are a special form of time-dependent function, where each state is dependent upon the prior states. Traditional methods for approximating them involve searching for trends and fitting known functions to them, or averaging prior results to attempt to guess the next ones. Determining the coefficients and orders of the variables is very difficult for these commonly very complex and often dynamically changing functions. Using an ANN, there is no need to start with an approximation of the time series function. ANNs are dynamically adaptable to online training, able to track and learn patterns and functions. Training an ANN as a time-series predictor simply requires using the time series result from t as a target for the result the ANN output at time $t-n$, where n is the desired prediction time.

System identification is a specialized form of time series prediction that tends to be highly dynamic and sensitive to a number of environmental conditions that require re-tuning in any traditional functional model of the system when the environment changes. The dynamic adaptability of an ANN enables it to not merely approximate the time series function, but to also track the function when it changes due to external conditions.

1.1.3. Shortcomings and Limitations. The first generation of ANN was limited by its requirement of binary inputs and outputs. While it was capable of universal Boolean function approximation, it required large numbers of neurons to be able to perform binary input and output of continuous values. The second generation upgrades this capability by replacing the thresholding function with an activation function that can output that varies across a continuum. This increased the calculation capabilities of a single neuron over the binary, digital output of the first generation considerably. Modern-day multi-layer perceptrons (MLPs) typically use second-generation neurons with sigmoidal activation functions like (1) rather than simple threshold gates.

$$f(x) = \frac{1}{\left(1+e^{-x}\right)} \tag{1}$$

The network architectures which evolved from the second generation include feedforward networks and recurrent neural networks (RNNs), as well as radial basis function (RBF) networks. These networks were capable of the same universal Boolean function approximation as their predecessors, but with fewer neurons required. Additionally, their ability to output across a continuum made them able to approximate analog functions as well [6], [20]. Second generation networks' power as learning algorithms was truly unlocked by their ability to support learning algorithms such as backpropagation [4], [5].

The problem faced by second generation ANNs is their inability to scale well. Just as the first-generation binary networks required far more neurons to approximate functions that second-generation ANNs can handle with relatively few neurons, there remain problems that second-generation ANNs find to have intractable computation requirements. As the number of inputs and outputs increases, the number of neurons required in a second-generation ANN increases even faster. When real-world system identification problems, such as wide-area monitoring for large-scale power systems [21], require tens to hundreds of inputs, it becomes simply beyond the capability of second-generation ANNs to perform the calculations in reasonable time with reasonable resources.

Living brains in living creatures, on the other hand, handle enormous amounts of information and perform numerous simultaneous highly-complex calculations, value judgments, and adaptive efforts all at once without a bit of slow-down. It behooves the neural network community to return to these original inspirations for the ANN to learn how they accomplish this feat, and use the insights gained to develop a third generation of neurons and neural networks that is to the second generation what the second generation was to the first.

The third generation is defined in [3] to be SNNs. It is on these third generation neuronal networks that this dissertation focuses.

1.2. RESEARCH OBJECTIVES

Despite the introduction of Ianella and Back's one neuron per possible output SNN that can perform arbitrary continuous-value input-to-discrete-output in 2001 [22] and Mass [23] and Duncombe's [24] works in modeling MLPs with spiking neurons instead of sigmoidal ones, there has been little effort in the area of continuous-valued SNN inputs and outputs. Using Inter-Spike Intervals (ISIs) as the encoded input values, Iannella and Back's method was capable of encoding any continuous value desired. Their output mechanism was to control which of the neurons in their feedforward SNN fired. However, because each neuron is trained to spike to represent one and only one output, this generates a step-function approximation of whatever arbitrary function the SNN is trained to approximate. Very discrete inputs or very large numbers of output neurons are thus needed to generate anything resembling a smooth output curve, which defeats any scalability benefits one might otherwise obtain from using spiking neurons. Developing a means of using continuous-valued inputs and producing continuous-valued inputs with arbitrary functional mapping is the key to truly bringing the SNN into its own as an heir to the second-generation ANNs.

Late in the last decade, Rowcliffe and Feng [25] produced the first successful attempt to use third generation ideas to produce an SNN which can operate on continuous-valued numbers and produce arbitrary responses. They abstract the actual spiking into a complicated activation function, allowing their neurons to operate in the same feedforward, feedback, and cellular structures as first- and second-generation neurons without any need for fancy encoding or decoding. The work of Mass [23] and Duncombe [24] demonstrated by Sharma and Srinvasan in [26] more resembles this in practice (albeit lacking the abstraction) than it does the biologically inspired SNNs (BSNNs) focused on in this dissertation.

By calculating an ISI from equations integrating expected responses to input firing rates that represent the continuous-valued inputs and never actually generating a spike train, they deliver an advancement comparable to that in [6]. This abstracted SNN (ASNN) may provide greater computational power than a second-generation counterpart, but can it match a more biologically inspired SNN (BSNN)?

The specific objectives of this research are to:

- review state-of-the-art encoding methods to pass continuous-valued inputs into an SNN
- determine whether the encoding methods proposed actually successfully pass information into the network
- review and utilize a state-of-the-art architecture that takes full advantage of the power of the spiking neuron model
- develop and test a decoder that can produce arbitrary functional mappings of the continuously-valued inputs into equally continuously-valued outputs
- test an abstracted representation of a spiking neuron in a traditional feedforward ANN structure and compare it on a real-world application to a second-generation ANN of the same structure that uses sigmoidal neurons
- demonstrate the abstracted and bio-inspired SNNs on a real-world problem of neuroidentification of generator dynamics in a multi-machine power system
- utilize a BSNN framework to perform neuroidentification of generator dynamics

1.3. CONTRIBUTIONS

Any exploration of the power and capacity of SNNs must be able to compare them to their predecessors. In order to accomplish this, it must be possible to test SNNs on the same sorts of problems as the second-generation networks. Despite the lack of continuous-valued output decoding efforts on BSNNs, there have been several efforts at creating methods for encoding continuous-value functions [27], [28], [29]. However, prior to [30], it had been left to assumption that the spikes generated by these methods truly contained the continuously-valued information passed into the encoding algorithm that generated the train.

This work seeks to correct this by testing several encoding methods and using their spike trains to reproduce the original input, thus proving the information exists in the spikes. It then goes on to encode data into a PSN and develop a decoding mechanism to translate the spiking outputs of the BSNN neurons into continuous-valued numbers.

Additionally, the ASNN of [25] is tested on a highly-complex real-world problem of power system identification, and compared to a time-delay MLP [31]. These demonstrations of both abstracted and biologically-modeled third-generation NNs and their computational power compared to second-generation NNs is the third major contribution of this dissertation. It is due to the greatest contribution of this work – the development of a framework for using BSNNs on continuous-valued functions – that finally enables the full fledged comparison of BSNNs to second-generation NNs and their development as a mature and functional computational tool.

The specific contributions of this dissertation are:

- examination, algorithm identification, and analysis of state-of-the-art encoding methods
- development of a novel new encoding method and associated algorithms
- development of algorithms for reversing the encoding methods examined, extracting the original data from the spikes to prove that the spike streams do carry the original continuously-valued data
- explanation, analysis, and execution of the Izhikevich model neuron in a polychronous spiking network (PSN), and maturation of the same
- development of a decoder capable of translating spikes from the PSN neurons into continuously-valued outputs which are tuned to isolate the functional representation desired
- demonstrate polychrony and its ability to vastly increase the capacity of an ANN
- testing of the completed PSN framework (encoder, PSN, decoder) on several problems, demonstrating the successful implementation of an SNN on continuous-valued inputs and outputs
- explain, implement, and compare the ASNN of [25] to a traditional MLP on a multimachine power system identification problem
- prove that a BSNN is capable of performing similar neuroidentification of generator dynamics – a continuous-valued real-world problem

1.4. SECTION SUMMARIES

1.4.1. Spiking Neural Networks.

1.4.1. Spiking Neural Networks. Section 2 begins the examination of SNNs in general. The state of the art in SNN modeling is examined along with their evolution from the first-generation ADALINE thresholding neurons and the Hodgkin-Huxley model through modern-day Izhikevich models. Discussion of the attempts in the last half-century to develop spiking neuron models as function approximators and as modeling mechanisms to study living brain dynamics follows.

The distinction between ASNNs and BSNNs is studied, and the strengths and weaknesses of both are examined in light of possible applications to which SNNs as a whole might be applied. Finally, the limitations that hamper the use of SNNs under the state of the art are examined, as these are the hurdles the work presented in this dissertation attempts to surmount.

1.4.2. BSNNs and Applications Thereof. In Section 3, the primary contribution of this work is outlined and detailed. A framework for a BSNN which can handle continuous-valued inputs and outputs is introduced, and encoding methods for transforming those inputs into spike streams which can stimulate a BSNN's dynamics are examined and tested to ensure they do not simply produce noise; the spike trains produced can be decoded to recover the original continuous-valued inputs.

In theory, any BSNN model which mimics brain-like spiking functionality can be used as the computational engine in this framework. In practice, the Izhikevich model has come to the fore in recent years as the go-to model for simple and accurate modeling of biological spiking neuron behavior. Arranged as a Polychronous Spiking Network (PSN), this is the model chosen for use in the work presented in this dissertation.

While encoding methods had existed prior to this work, and the PSN has been known for half a decade, this dissertation presents what is, to the author's knowledge, the first successful means of decoding continuous-valued outputs from a BSNN and training them to a target function.

1.4.3. Neuroidentification of Power Systems with SNNs. The real-world application chosen to demonstrate the capabilities of SNNs is power system identification. Predicting the speed and voltage deviation of generators is a highly complex problem which has time-dependent dynamics in its functional form. Section 4 demonstrates the application of SNNs to this difficult problem, and compares their performance to an MLP. Additionally, a BSNN is demonstrated to be able to learn to identify generator dynamics, proving that BSNNs are capable of continuous-valued outputs on complex nonlinear functions.

1.5. SUMMARY

This dissertation presents a review of NNs leading to the development of SNNs and details the development of means of using them on the same sorts of problems as previous-generation NNs. It goes on to compare the performance of SNNs to their predecessors on several problems, and apply them to a real-world system identification problem which faces scaling issues when second-generation NNs are used on them.

Having enabled SNNs to be used and tested against prior-generation NNs on any sort of problem on which NNs are currently used or benchmarked, this dissertation opens the way for SNNs to truly contend for their claimed position as the third generation of NN.

2. SPIKING NEURAL NETWORKS

2.1. INTRODUCTION

Neural networks have been studied for more than half a century, though they were popularized after Werbos introduced the world to backpropagation training as a means of making them self-adapting learning machines [4], [5]. They have evolved from simple MLPs to increasingly different and complex feedforward [32], feedback [32], and cellular forms [33], and the perceptron neuron has had variants and adaptations. Some are attempts to capture different neuronal behaviors, while others are efforts to develop better computational engines [3].

As understanding of the brain – the ultimate inspiration for neural networks – increases, efforts to develop more brain-like neuron structures and network architectures in order to capture more of the brain's powerful computational capacity and efficiency are underway [34]. Spiking neurons and networks are the primary focus of this work, and both abstracted perceptron-style neurons with activation functions focused on spiking behavior as well as biologically-inspired models of voltage-accumulating-and-firing neurons (particularly the Izhikevich model) are explored.

The ultimate goal is to develop spiking neural networks (SNNs) to the point that they can perform comparable functions to their predecessors (such as the MLP), and then test them on problems which their predecessors found too intractable, such as high-input, high-output neuroidentification, e.g. of large-scale wide-area power systems.

To achieve that goal, one must first understand the state of the art in SNNs. Section 2.2 gives an overview of SNN research, including various models and implementations as well as prior work on abstracted SNNs as an advanced activation function for function approximation and the Izhikevich model of biologically-inspired spiking neurons. Section 2.3 goes into applications to which SNNs have been applied, while Section 2.4 outlines the limitations of current work in the area and thus where work still needs to be done.

2.2. STATE OF THE ART

Spiking neural networks (SNNs) have been a subject of interest since the 1950s and the Hodgekin-Huxley model [2], developed by biologists studying a giant squid's neuron. Left by the wayside along with all NNs before [5] introduced a means for the first generation of NNs to be used effectively as problem solvers beyond the scope of the McCulloch-Pitts neuron and ADALINE, a single-layer neural network.

In the 1990s, the idea of spiking neurons began to draw renewed interest. Called a third generation of neural network in [3], spiking neural networks seek to advance the activation function at least as much as the second generation did by replacing thresholding functions with continuous-valued ones. Though [2] outlines the biological neuron's functionality in great detail, the earliest mathematical model was the integrate-and-fire proposed by Lapicque, which modeled a neuron as a resister and capacitor in parallel [1]. Despite having no knowledge of the biological structure of neurons at the time of its creation at the dawn of the last century, this model captures enough of the actual function of biological neurons to still be in wide use today.

Spiking neurons are hoped to represent as great a leap forward in computational power as were continuous activation functions, but they pose a unique set of problems. Where ADALINE and other thresholding functions operated strictly on binary inputs and outputs, continuous activation function neurons can operate directly on continuous-valued inputs and directly produce continuous-valued outputs. Spiking neurons, on the other hand, can only operate on voltage-level inputs (and almost always are designed, as their biological inspiration, to do so in the form of spikes) and can only output voltage spikes. This could be seen as a noisy binary input/output, and, indeed, spiking neurons and small feed-forward networks of the same have been used to generate XOR gates [10], [11], [12].

However, even should spiking neurons prove to be as much of a step forward in computational power on Boolean problems as were continuous activation function neurons compared to thresholding functions, they remain a step backwards if one cannot utilize them for continuous function approximation. Many different encoding and decoding schema have been considered for translating continuous-valued numbers into spikes and back again. Many rely on either spike rate/frequency or ISIs.

One of the earliest ISI-based encoding methods was presented in [22], wherein the input is coded as the time between two spikes (the ISI). This provides an arbitrarily continuous precision on what input values it may handle. The one-layer feedforward network of spiking neurons is then trained so that each spiking neuron responds to a particular range of ISIs, spiking if its particular ISI is fed to the network. The other neurons' outputs are suppressed by that same input. Each neuron thus learns one particular output for a range of continuous inputs, and the network can map an arbitrary function. Unfortunately, this sort of encoding and decoding mechanism generates only as many possible outputs as there are neurons, creating a highly discrete step-function that does not output continuous values at all.

One idea to allow spiking neurons to operate in continuous number domains is to abstract them into a kind of continuous-valued activation function, which would permit them to simply "upgrade" the second generation neural models as they "upgraded" the first [25]. The work in [23]-[24] uses ISIs for the outputs rather than abstracting them and achieves better results than [22], but like [25] is still a feedforward architecture which does not capture the inter-neural structure of the SNN, and so exploration of more biologically-faithful models of both the spiking neuron and the network architectures in which they serve as the functional units continues.

It is natural that these various models of SNNs should be tested to demonstrate their capabilities, and so problems on which the spiking nature of the inputs and outputs have been found. Image and video processing [12], [13], as well as pattern-matching and classification [10], [11], [12], have been the standard problems to which SNNs have been applied and on which they have been tested. More ambitious experiments include attempting to classify inputs according to other sensory stimuli other than sight (auditory [18], gustatory [19]).

The SpikeCell neural model presented in [14] utilizes a deterministic spiking neuron to attempt to improve on the standard static neurons of the second generation, while emulating the outputs of neurons with more traditional activation functions. Using an internal potential $V(t)$ which evolves in time according to (2) and a dynamic threshold $V_{thresh}(t)$ which evolves in time according to (3). SpikeProp neuron i can emit positive or negative spikes as delta functions $\delta_i(t+1)$ defined by (4). For the digital discretization, d

represents a step size that allows this model to be tuned to a desired level of precision in emulation.

$$V_i(t+1) = V_i(t) + d\sum_j w_{ij}\delta_j(t) \tag{2}$$

$$V_{thresh_i}(t+1) = V_{thresh_i}(t) + d\delta_i(t) \tag{3}$$

$$\delta_i(t+1) = \begin{cases} -1 & \text{if } V_i(t) < V_{thresh_i}(t) \\ 0 & \text{if } V_i(t) = V_{thresh_i}(t) \\ 1 & \text{if } V_i(t) > V_{thresh_i}(t) \end{cases} \tag{4}$$

The threshold voltage V_{thresh} of neuron i is thus dependent on the value of the delta function spike δ_i it fired in the most recent time step. The positive and negative spikes output by these neurons are really thresholded functions similar to first-generation perceptrons, and SpikeProp has been successfully used on classification experiments similar to those for which that first generation were demonstrated to be successful.

More computationally-minded models tried lesser or greater complexity to capture simply the behavior of a spiking neuron or to experiment with more biologically faithful mechanisms which might lead to that behavior naturally. A neurogenetic model presented in [35] focuses as much on molecular research as it does on actual SNN applications. By evaluating the Local Field Potential (LFP) of a test neuron or network thereof, the neurogenetic model is compared to a target electroencephalogram, or EEG. The actual method of generating the spiking neurons is similar to a Genetic Algorithm (GA) which utilizes a number of potential proteins to build each neuron. Careful modeling of the chemical processes that each protein performs generates the LFP which can be compared to an EEG. While this is not helpful overall in terms of applications, it is an interesting model for those who wish to simulate brain behaviors.

The usefulness of SNNs as models of living neuronal networks has not gone unnoticed in the neurobiology and neuroengineering communities, either. Increasing amounts of collaboration between the computer engineering and neurobiology fields has led to work on interfacing living neurons with computer systems and even allowing one to control the other [36]. The multi-electrode arrays (MEAs) introduced in that paper are used at the Georgia Institute of Technology to interface rat brain cultures with computers, so their spiking behaviors can be stimulated and monitored [37]. There is even a robotic arm that can be connected over the internet to the system at Georgia Tech which receives stimulus from a camera on site and controls the robotic arm to doodle and scribble in response [38].

In part because of the increased ability to monitor biological neurons and their patterns, more efforts such as [39] emerged as the decade advanced. The digital spiking neuron (DSN) and quantized spiking neuron (QSN) were used to generate discrete "spike position maps," which can be used for pattern identification, though no real applications are attempted.

In the same year, however, [12] came out with the SpikeProp model of spiking neurons which uses a form of temporal encoding originally proposed in [28]. Section 3 will go into more detail on the Gaussian receptor field (GRF) encoding method presented in that work, but [12] developed a form of backpropagation-based weight learning. SpikeProp is a feedforward set of spiking neurons, and successfully learned the XOR gate function. One of the problems they tackled was the need to have their backpropagation have a special case for times when the neurons do not fire, because spiking neurons do not always output a value at each iteration. For decoding, SpikeProp simply assigns an early firing time to logical value "1" and a late firing time to logical value "0," and ensures that the converged neural network fires at least once per time-window for any input.

Perhaps ironically, a model of spiking neurons that uses the same "behavior-focused" rather than "method-focused" sort of design demonstrated in [1] is one of the most widely-acclaimed SNN models today. Introduced in [40] by a neurobiologist, the Izhikevich model of spiking neurons (explained in more detail in Section 2.2.2) is known to very well simulate actual brain behaviors [41], [42].

2.2.1. Abstracted Spiking Neural Networks (ASNNs). One approach to incorporating SNNs into standard NN applications is to try to maintain the input/output architectures already well-known and commonly practiced. MLPs, Radial Basis Function networks (RBFs), Recurrent Neural Networks (RNNs), and most other commonly-used architectures have a layered structure with neurons which operate on real-world-relevant values in activation functions and pass them on to the next (usually output) layer.

An excellent example of one such effort is performed in [25]. The ASNN developed in that work utilizes an abstraction of the ISI based on the mean and standard deviation of hypothetical spikes fired in response to the real-valued input. Figure 2.1 illustrates the abstracted spiking neuron.

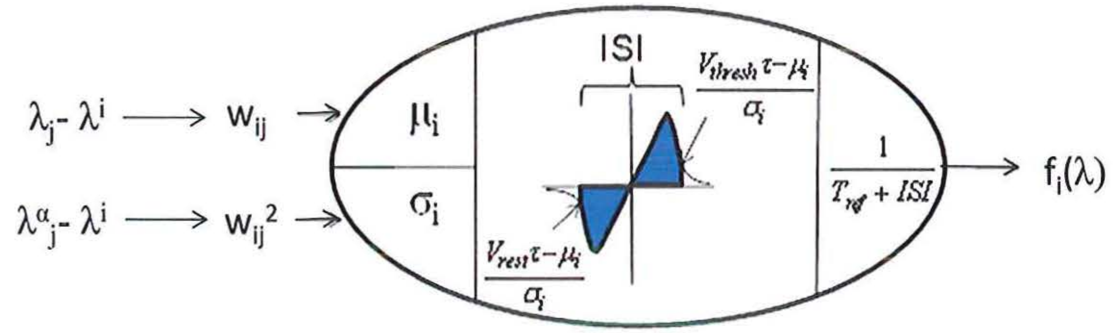

$\lambda_j - \lambda^i$
w_{ij}
$\lambda^{\alpha}_j - \lambda^i$
w_{ij}^2
μ_i
σ_i
ISI
$\frac{V_{thresh}\tau - \mu_i}{\sigma_i}$
$\frac{V_{rest}\tau - \mu_i}{\sigma_i}$
$\frac{1}{T_{ref} + ISI}$
$f_i(\lambda)$

Figure 2.1 Abstracted spiking neuron serving as an advanced sort of activation function.

The mean μ and standard deviation σ for each neuron i are based on the temporal distribution of spikes which would be firing if they were not abstracted away. They are defined by (5) and (6).

$$\mu_i = \sum_{j=1}^{n} \lambda_j - \lambda^i \;\; \omega_{ij} \;\; 1-r \tag{5}$$

$$\sigma_i^2 = \left(\sum_{j=1}^{n} \lambda_j^{\alpha} - \lambda^i \;\; \omega_{ij}^2 \right) 1+r \;\; + \rho \sum_{j \neq h=1}^{n} \left(\lambda_j^{\frac{\alpha}{2}} - \lambda^i \right) \omega_{ij} \omega_{ih} \;\; 1+r \tag{6}$$

The j^{th} input into the neuron is given as λ_j while α is a tuning constant that must be greater than 0. The ratio of excitatory to inhibitory inputs is given by r, and is usually set to 1 for an equal number of each. This causes μ to typically be zero, so the experiments in [31] derive all the meaningful input from the standard deviation of the spikes. The superscripted λ^i values represent centers in the calculation space. Each neuron i has a different center around which it responds, and those centers are found in any of the same ways that RBF network centers can be found [25].

This, however, merely transforms the real-world input value given by λ into an abstracted temporal spiking behavior. The model in [25] and [31] uses the abstracted firing rate as calculated using the ISI as the final output from the activation function. In order to achieve this, the ISI first is calculated by means of (7), with V_{rest} as the resting voltage of the spiking neuron, and V_{thresh} as the threshold voltage. The relaxation period of the neuron, τ, is the time it takes for a given spike's influence on a neuron to fade.

$$ISI = \frac{2}{\tau} \int_{\frac{V_{rest}\tau - \mu_i}{\sigma_i}}^{\frac{V_{thresh}\tau - \mu_i}{\sigma_i}} g(x)\mathrm{d}x \tag{7}$$

Integrating over the Dawson's Integral *g(x)* given in (8) between the limits established by these voltage values and the mean and standard deviation of the abstracted spikes passing through the spiking neuron gives us the ISI.

$$g(x) = e^{x^2} \int_0^x e^{-u^2} \mathrm{d}u \tag{8}$$

The final piece of the firing rate that is used as the output of this abstracted spiking neuron is the refractory period T_{ref}, which is the time after a neuron spikes that it spends below its resting voltage before it finally recovers and resumes its resting voltage. The firing rate is the multiplicative inverse of the sum of T_{ref} and the ISI (9).

$$f_i(\lambda) = \frac{1}{T_{ref} + ISI} \tag{9}$$

This highly complex procedure can produce complicated functional surfaces. The relatively simplistic one produced with r=1 and α=2 is shown in Figure 2.2.

This form of abstracted spiking neuron can be used in place of traditional sigmoidal neurons or RBF neurons in feedforward, feedback, and recurrent architectures. As Section 4.3.2 will demonstrate, this is a powerful form of neuron despite remaining more akin to traditional neuron architectures than biological models of spiking neurons.

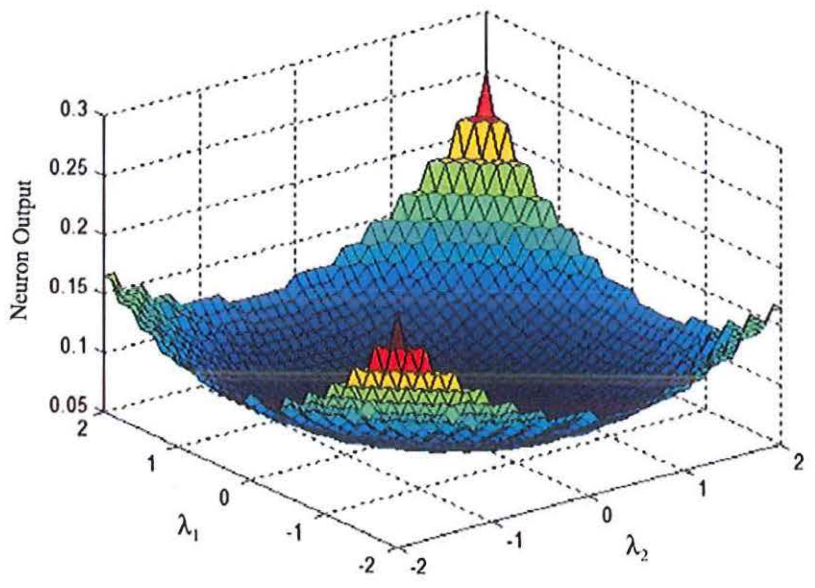


Figure 2.2. Output of an abstracted spiking neuron with $r = 1$ and $\alpha = 2$, with λ_1 and λ_2 varying independently from -2 to 2 with a step size of 0.01.

2.2.2. Biologically-Inspired Spiking Neural Networks (BSNNs). Biological neurons are very complicated machines with numerous mathematical models to approximate their behavior. The oldest of these is the Hodgekins-Huxley model [2], which has many equations modeling very specific behaviors observed empirically in the dissection of a giant squid neuron. Ion channels, conduction paths, axons, dendrites, and many other components of the biological neuron were studied and modeled.

The generally-accepted approximations of the biological neuron focus primarily on the voltage transfer between them. Neurons have three generally-important components to modeling this behavior: dendrites, which collect voltage spikes through synapses which connect them to other neurons; the main neuron body (or "soma"), which is where the charge is stored while it builds; and the axon, which is the output path along which a voltage spike is released. The "leaky neuron" model sees voltage spikes from pre-synaptic neurons flow into the dendrites of a neuron, and build up the voltage stored on the soma. This voltage leaks away naturally over time, but if multiple spikes come in over a short enough period, the voltage builds faster than it leaks away. When it reaches the threshold voltage V_{thresh}, it emits a spike of its own. The soma's voltage immediately drops below its resting voltage V_{rest}, and the spike propagates along the axon to synapses which connect to other neurons' dendrites.

The Izhikevich model of neuronal activity is a simple two-equation system, (10) and (11) that represents the voltage over time of a spiking neuron [40].

$$\begin{cases} v(k+1) = 0.04v(k)^2 + 5v(k) + 140 - u(k) + I & \text{if } v(k) < 30\,\text{mV} \\ v(k+1) = c & \text{if } v(k) \geq 30\text{mV} \end{cases} \tag{10}$$

$$\begin{cases} u(k+1) = a(bv - u(k)) & \text{if } v(k) < 30\text{mV} \\ u(k+1) = u(k) + d & \text{if } v(k) \geq 30\text{mV} \end{cases} \tag{11}$$

The voltage at discrete time-step k is given by $v(k)$. This is the value of the voltage assumed to rest in the soma of the simulated neuron. It builds due to the input from other neurons represented by I, and decays with time (if it does not spike). The threshold voltage is set at 30 mV. After a spike, the voltage resets to c, which (along with a, b, and d) is one of four variables which can be tuned according to Figure 2.3 to cause the neuron to behave according to one of several ways real biological neurons tend to spike. The experiments run for this dissertation all use regular spiking excitatory and fast spiking for inhibitory neurons in any Izhikevich model experiment.

The neural model alone faithfully represents the input and output voltage spike behavior of biological neurons, but the most potentially ground-breaking advance is in the polychronous network introduced in [43]. This polychronous spiking network (PSN) develops "polychronous clusters" of neurons which respond only when specific neurons are triggered in specific orders, as shown in the example five-neuron network with delays as displayed in Figure 2.4. The delays align such that incoming spikes arrive all at once if the pre-synaptic neurons are triggered in specific orders, but do not if they are triggered out of sequence or with the wrong timing. Section 3.4 goes into it in more detail as it discusses details of the PSN itself.

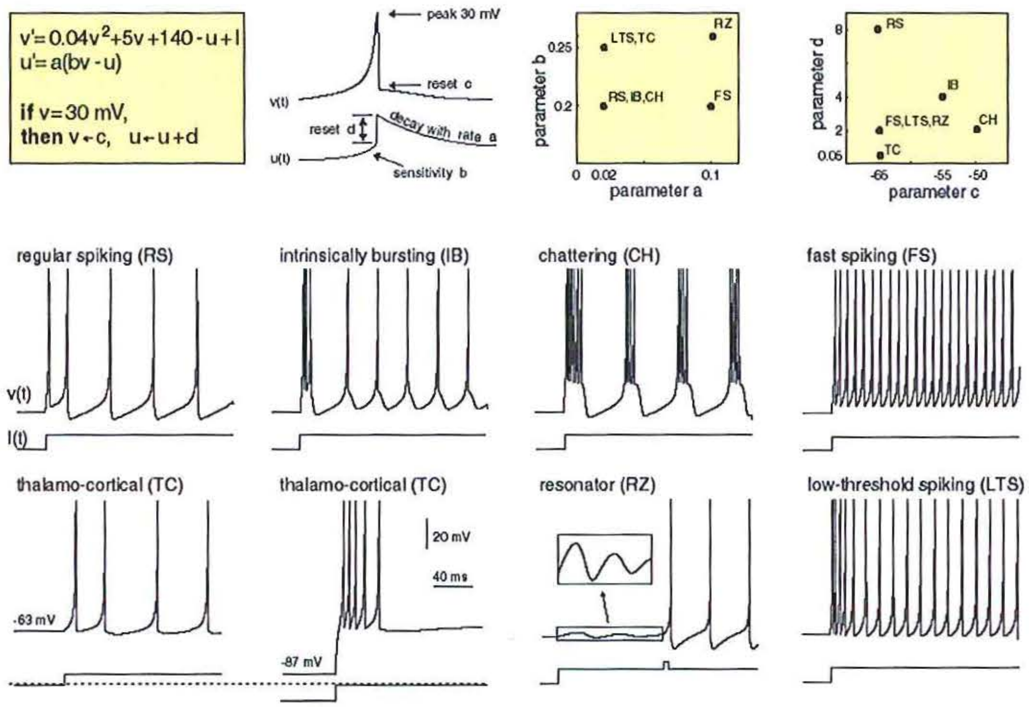


Figure 2.3. Diagram of possible spiking behaviors of the Izhikevich model artificial spiking neuron, along with the settings required to achieve each variety. *Electronic version of the figure and reproduction permissions are freely available at* *www.izhikevich.com*

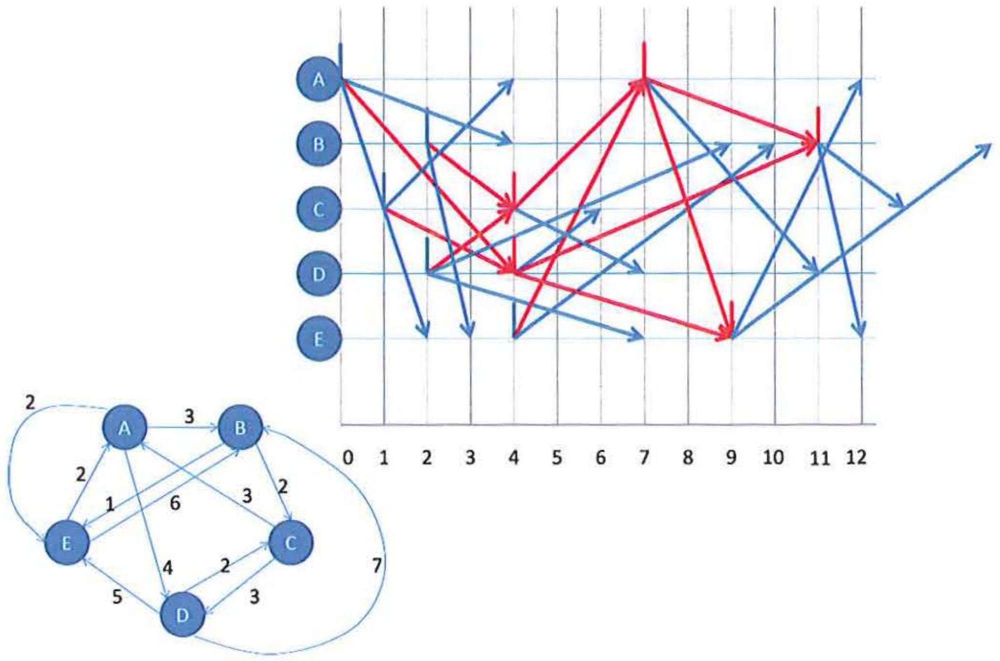


Figure 2.4. Example of polychrony in action. Blue spikes are inputs. Red lines are synaptic links which coincide from stimulus spikes to cause post-synaptic spikes. The delays on the connections between neurons are shown in the five-neuron diagram.

Such clusters can each represent a mapped output, and each additional neuron to the PSN increases the number of clusters exponentially. Thus, for each new Izhikevich neuron added to the PSN, increasing numbers of additional possible outputs become available. This is extremely similar to living brains, which exhibit literally astronomical calculation capacity without needing more neurons than there are stars in the sky.

2.3. APPLICATIONS

SNNs are just now beginning to leave infancy and toddle around the world of real applications. They are skilled pattern recognizers, and there have been a number of successful spiking neuron-based logic gates, such as the XOR gate demonstrated in [10], [11], and [12]. It is desired that SNNs be useful in as many functions as their ancestors. In their broadest definition, neural networks are function approximators. This means that any application where a current state leads to a new state via some function can use a neural networks solution. The first generation perceptrons were capable only of universal Boolean approximation, but the second generation's introduction of continuous activation functions expanded this to any analog or digital function. More realistically, however, neural networks are most useful when the function transforming one state to another is highly complicated, poorly understood, changing with time, or some combination of these circumstances, because in simpler situations a straightforward analytic function would be easier to generate and utilize.

SNNs, as they stand today, are commonly tested on pattern-matching and image processing problems. Classification of data sets is also common [7]-[19]. However, the biggest advances have been in attempting to use them as artificial, fully-monitorable models of living brains for neurobiological study [40], [43].

In order for SNNs to earn their position as a "next generation" neural architecture, they must prove capable of matching their predecessors at this general task of handling real-world inputs and outputs in continuous regimes. From there, the hoped-for increased computational power and versatility of the SNN will enable neural networks to be used on applications too computationally intractable for current-generation architectures. Because of the similarities between the behavior of the Izhikevich SNN and the MEAs in [36], it is hoped that successful implementation of this third generation of artificial neural network will also enable more powerful and accurate computation with living neurons as the processors.

The SNN's basis in the way biological brains process information gives hope that it will invert the scaling problem faced by MLPs, RBFs, RNNs, and the like. Ever-increasing numbers of hidden neurons are required in first- and second-generation NNs to handle the special cases and intricate dynamics of the function to be approximated as the number of inputs and outputs increases. This is a geometric to exponential rise, and makes many applications – such as the wide area monitoring system proposed in [21] if scaled up to handle the New England or Brazilian power grids, as proposed in [44] – intractable. Living brains monitor and control vastly more complicated machines (such as mammalian bodies) without facing such problems. The third generation of NNs is designed to capture this.

2.4. LIMITATIONS OF EXISTING WORK IN SPIKING NEURAL NETWORKS

Achieving the kinds of continuous-value function approximation required by real-world time series and neuroidentification problems requires means of inputting continuously-valued real-world data into the SNN, and retrieving meaningful continuously-valued data from the SNN's outputs. The state of the art for SNN applications tends to focus on video processing, pattern matching, and very limited input and output mechanics (usually comprising a limited set of patterns or values which can be read in and out).

There are many different methods for inputting data into SNNs, but no good means of retrieving continuous-valued outputs from them. Moreover, prior to the encoding work presented in Section 3.3, the few methods that do exist for converting arbitrary numeric values into spike streams for input into an SNN are simply assumed to contain the encoded information. With no means of decoding the SNN's spike patterns, the information was never checked. This dissertation investigates the spike streams created by three encoding methods to determine if the information is present in the spikes, and goes on to develop a possible decoding method to obtain meaningful real-world values from the output of an SNN.

2.5. SUMMARY

Spiking neural networks drink more deeply from the font of inspiration out of which neural networks in general were conceived. By more carefully modeling the actual behavior of biological neurons, they seek to capture the calculation power and efficiency of those natural computers to overcome problems – most particularly problems of database scale – which living brains do not even notice but on which current-generation neural networks tend to choke. Neuroidentification of large-scale power system dynamics requires not only massively scalable architectures, but also the ability to operate on continuous data spaces rather than being restricted to limited sets of patterns or other specified inputs and outputs.

The next section goes into more detail on the Izhikevich model of a spiking neuron and how it can be used in a new biologically-inspired dynamic reservoir. For it to be truly useful in such neuroidentification and function approximation problems, however, a means of ensuring the real-world data is present in the encoded input spike streams is essential. Moreover, a decoding mechanism for extracting the calculated spike coded outputs into continuously valued meaningful numbers is required before it can even match its predecessors' performance, let alone demonstrate its superiority in large-scale situations.

3. BSNNS AND THEIR APPLICATIONS

3.1. INTRODUCTION

The major contribution of this dissertation is the development of techniques for applying SNNs to all of the same sorts of problems that second-generation NNs can handle. It is essential to either prove that SNNs can do anything that second-generation NNs can, or to determine where second-generation NNs are superior, before SNNs can be properly developed as a third-generation NN.

Because ASSNs and the feedforward networks utilizing spiking neurons resemble second-generation NNs, they are already capable of mimicking second-generation functionality (as explained in Sections 2.2.1 and 4.3.2). The main problem lies in BSNNs and their peculiarities that make handling continuous-valued numbers difficult-to-impossible. Presented here is a framework for using a PSN as an engine for a third-generation neural network. This framework is, to the best of the author's knowledge, the first to make general continuous-number computation with BSNNs possible. The overall framework is explained in Section 3.2. Section 3.3 explains and demonstrates state-of-the-art encoding methods for transforming arbitrary continuous values into spike patterns, then goes on to confirm that all three methods generate spike patterns which actually contain the encoded information. Section 3.4 describes the PSN itself and how it works; the engine that actually performs the computational "heavy lifting" for the BSNN. Decoding, essential to finish the cycle and recover meaningful continuous-valued numbers from the calculated dynamics of the PSN, is discussed in Section 3.5, and example problems on which this framework has been tested are given in Section 3.6.

3.2. BSNN FRAMEWORK

The SNN's processing capabilities are useless without a means of inputting data on which it is to operate, and a means of extracting the processed information in meaningful forms. This requires a framework like that in Figure 3.1, with an encoder to translate real-world values into spike trains and a decoder which converts the SNN's spikes to meaningful real-world values [29].

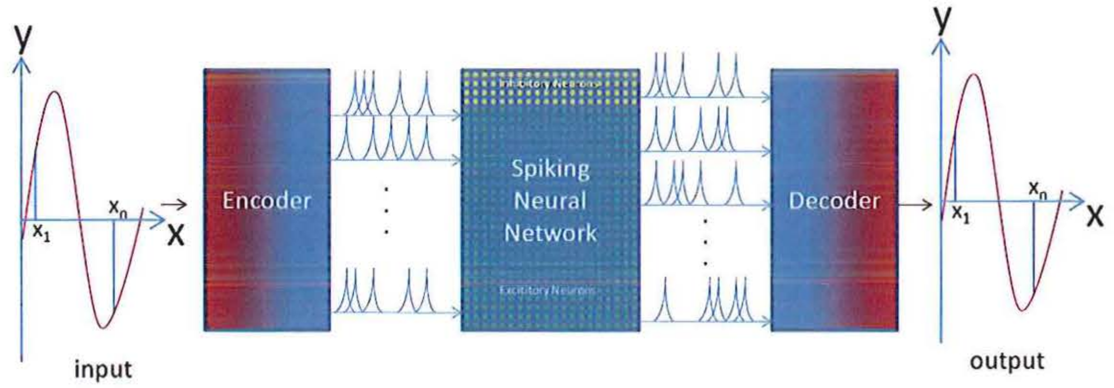


Figure 3.1. Block diagram of a SNN used to fully reproduce a time series of continuous-valued data.

The various sub-modules of the blocks are explained in the following sections. The SNN itself does the heavy mathematical lifting, and the decoder translates its spike responses into real-world values and potentially tunes the outputs to meaningful targets.

3.3. ENCODING

For BSNNs to be as useful as their predecessor architectures, they need to be able to take in real-world values. While SNNs have been successfully applied to a selection of specific classification problems (of which, really, Boolean logic discrimination and image processing are subsets) such as those presented in [11], [28], one of the two biggest obstacles to using SNNs for all the functions that second-generation NNs could perform has been encoding real-world values into forms with which the spiking neurons could do anything. This is a problem for which solutions have been searched for years, as demonstrated in Section 2.1. Where handling continuous values is fairly trivial for MLPs, RBFs, and other second-generation NNs, it requires special encoding methods in order to convert numbers into spike trains for BSNNs. Iannella and Back [22] proposed a one-neuron-per-possible-output algorithm a decade ago. The ISI-based encoding method presented in their work is tempting, but not terribly robust to noise from the highly-recurrent system that makes up the PSN.

Three means of encoding and decoding spikes for spiking neural networks are examined here as stand-out methods of encoding real-valued numbers into the third generation NN. In particular, a Poisson rate encoding method [27] (called PREM by the authors of [30]) and a method based on GRFs [28] are analyzed, and an in-house dual-neuron *n*-bit representation (DNNR) [29] is presented. Each of these methods generates streams of spikes with different temporal or spatial patterns when fed continuous values over the range they are designed to handle. For a serious study of BSNNs as a whole, however, certainty is needed that the real-world values actually are encoded. That is, it is essential to be certain that the values encoded are contained and present as information in the spike trains generated by the encoding methods tested. The ISI method in [22]-[24] does obviously meet this requirement, at a minimum: the encoding method is so simple that it cannot fail to do so, and a simple stopwatch can recover the values by observing time between the spikes. PREM, GRF, and DNNR, however, need to be tested, to see if they can meet this minimum requirement before they can be held up in comparison of the ISI method at all.

This section not only, therefore, outlines the three named encoding methods, but then presents a reversing algorithm for each encoding method. If the spike streams generated contain the information encoded by the algorithm, the revering algorithm can take the spike stream and reproduce said values with no prior information about what the original values were.

3.3.1. Poisson Rate Encoding Method. Perhaps most closely related to the ISI method of encoding by its relation of firing rate (how often a neuron fires in a given period of time) to the real-world value to be encoded, the PREM creates a spike train whose pattern obeys a homogenous Poisson process for a given input value.

Characterized by (12), a Poisson process characterizes the probability that the number of events counted between times a and b is equal to k [45].

$$P\left[\left(N(b)-N(a)\right)=k\right]=\frac{e^{\lambda_{a,b}}\left(\lambda_{a,b}\right)^{k}}{k!}k=0,1,2,\ldots \tag{12}$$

The homogeneous process described in (12) assumes that $\lambda_{a,b}$ is constant. For a given real-world input value, this remains the case in the PREM. However, this encoding method uses the real-world input value as λ, which means that the Poisson process is overall inhomogeneous. The modification is simple, redefining $\lambda_{a,b}$ according to (13).

$$\lambda_{a,b}=\int_a^b \lambda(t)dt \tag{13}$$

While a straight-forward approach could simply generate a stream of spikes which have a probability of spiking equal to $\lambda_{a,b}$ in each time-step, [27] instead incorporates it into the biologically-inspired spiking neuron equations by incorporating $\lambda_{a,b}$ into the time-dependent voltage equation (14) as part of the mean and standard deviation of the distribution of the voltage spikes as shown in (15). Either method results in a string of temporally-random spikes with the information solely contained in the number of spikes input in a given period of time.

$$\mathrm{d}V=-\frac{V}{\gamma}\mathrm{d}t+\mu\mathrm{d}t+\sqrt{\sigma}\mathrm{d}B_t \qquad V\geq V_{threshold} \tag{14}$$

$$\begin{cases}\mu=a\lambda(t)(1-r)\\ \sigma=a^2\lambda(t)(1+r)\end{cases} \tag{15}$$

The ratio of inhibitory to excitatory neurons is, again, r, and γ is the product of λ and $V_{threshold}$. The use of μ and σ to transmit data is similar to that used in Section 2.2.1 and [25]. The always-positive magnitude of the excitatory postsynaptic potential is a, and B_t is a Brownian motion variable.

The mathematics behind the maximum likelihood estimation (MLE) method of transforming the spiking rate of the neurons governed by (14) is given in exhaustive detail in [27]; this dissertation will simply touch on the high-level theory in demonstrating that information encoded by the PREM is present in the output spike stream. Algorithm 1 provides the broad steps for enacting the PREM. Reference to [27] is still recommended for specifics.

The MLE method of recovering the inputs requires a Monte Carlo simulation of multiple possible "paths" for the neurons' firing patterns to follow. A "large number" of such paths is used in [27] in order to get a good simulation. PREM has a lower bound on possible input values, below which MLE returns "0" as the answer when it attempts to recover them. This lower bound approaches zero asymptotically as the number of paths approaches infinity. However, each additional path is, computationally, equivalent to an extra neuron. Because the other methods of encoding tested in this section operate on the order of a dozen neurons, the experiment in [30] uses twelve neurons with three paths each, for a computational cost equivalent to 36 neurons total. Algorithm 2 provides a step-by-step process for enacting the MLE method.

Algorithm 1 Encoding using PREM [27]

```
1: Initialize number of neurons $m$ and number of independent paths $B$
2: Determine the minimum threshold of possible inputs based on $B$.
3: Add the lower bound to all inputs in preprocessing
4: for each neuron $i$ do
5:     for each independent path $j$ do
6:         Use $\lambda(t)$ in (15) as the continuous-valued input
7:         Determine $dV$ in (14), and update neuron for next time step
8:     end for
9: end for
```

Algorithm 2 Recovering inputs from PREM [27]

```
1: for each neuron $i$ do
2:     for each independent path $j$ do
3:         Determine the rate of fire for path $j$ in neuron $i$
4:     end for
5:     Use the Monte Carlo averaging technique to estimate the rate of
       fire $\lambda$ of neuron $i$
6: end for
7: Estimate the maximum likelihood value using the MLE method in [27];
   this is the original continuous-valued input
```

With three paths per neuron, it turns out that only two paths ever fire at all on the sinusoid given in (16). The lower bound below which the MLE cannot resolve anything but "0" is roughly 1500 when only three paths are used, so a linear offset of 2000 is used to push (16) above this threshold. As Figure 3.2 illustrates, MLE can recover data encoded by PREM with only a -0.5 bias, which is constant and thus can be removed in post-processing along with the linear offset to compensate for having only three paths per neuron.

$$f(t) = \begin{cases} 20\sin(2\pi t) & 0s \le t < 1s \\ 20\sin(4\pi t) & 1s \le t < 2s \\ 20\sin(2\pi t) & 2s \le t < 3s \end{cases} \tag{16}$$

Uniquely amongst the encoding methods presented here, PREM and MLE actually pass the information into the neurons themselves before spike patterns are ever generated. If the authors of [27] ever develop a decoding method that does not strictly reproduce the inputs, this could be a useful tool for applying BSNNs to real-world problems involving continuous valued inputs and outputs.

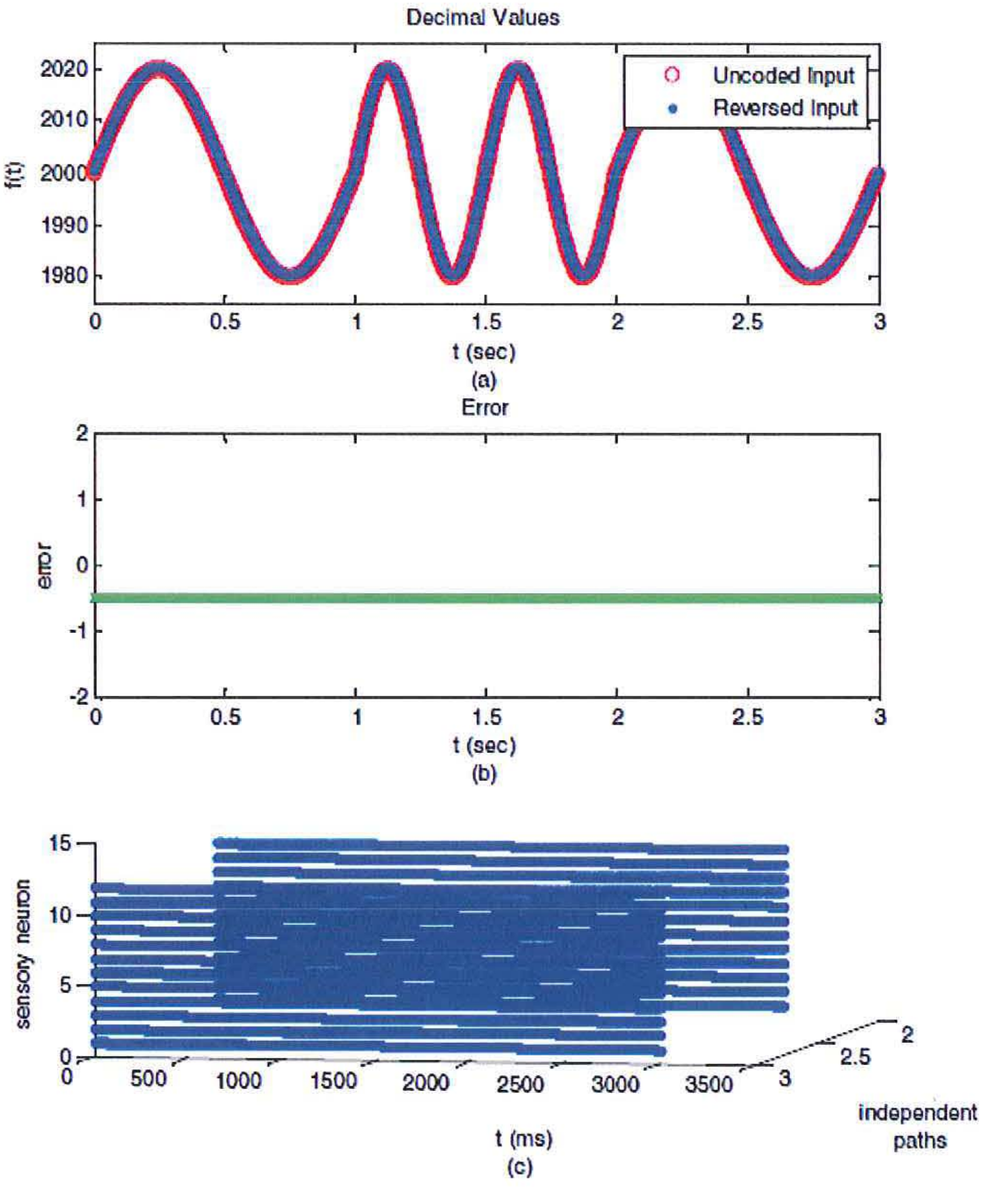


Figure 3.2. PREM reversibility demonstration. (a) Sine waves of 1 Hz and 2 Hz, original input and recovered values from MLE of PREM-encoding. (b) Error between original input and recovered values; note that it is a constant -0.5 bias. (c) Neuron spiking patterns on the 12 neurons with three paths each; one path never fired at all.

3.3.2. Gaussian Receptor Fields. Overlapping GRF are used to encode continuous real data into spatially and temporally defined spike trains [28]. Figure 3.3 provides a schematic diagram of the means of performing this encoding, which uses a

Gaussian activation function for each input (or sensory) neuron, with centers spread evenly over the possible continuous input range. For a range [n_{min}, n_{max}] of a variable n with m sensory neurons, the centers C_i for neuron i are determined by (17) and their widths w by (18). The activation value f_i for each neuron i is determined by (19).

$$C_i = n_{\min} + \frac{2i-3}{2} \times \frac{n_{\max} - n_{\min}}{m-2} \qquad m > 2 \tag{17}$$

$$w = \frac{n_{\max} - n_{\min}}{\gamma(m-2)} \qquad m > 2 \tag{18}$$

$$f_i(x) = Ae^{-\frac{(x-C_i)^2}{2w^2}} \qquad i = 1,2,\ldots,m \tag{19}$$

A single input value passed into all sensory neurons' activation functions will generate different activation values; the closer the input value is to the center of a given sensory neuron, the higher its activation value on that neuron. These activation values are translated via (20) into firing times inversely proportional to the activation value.

$$T_i = (1 - f_i) \times \tau \qquad i = 1,2,\ldots,m \tag{20}$$

The activation value for neuron i is given by f_i and τ is a constant that defines the maximum time delay possible. For the experiments in [30], τ is set to be 10 ms, and any neuron whose delay time T_i is greater than nine milliseconds is discarded as too weak to count for the given input. Thus, any given input value will cause two or three neurons to fire, depending on where it falls on their activation functions. The closer to the sensory neuron's center, the sooner it fires. Algorithm 3 explains this encoding process programmatically.

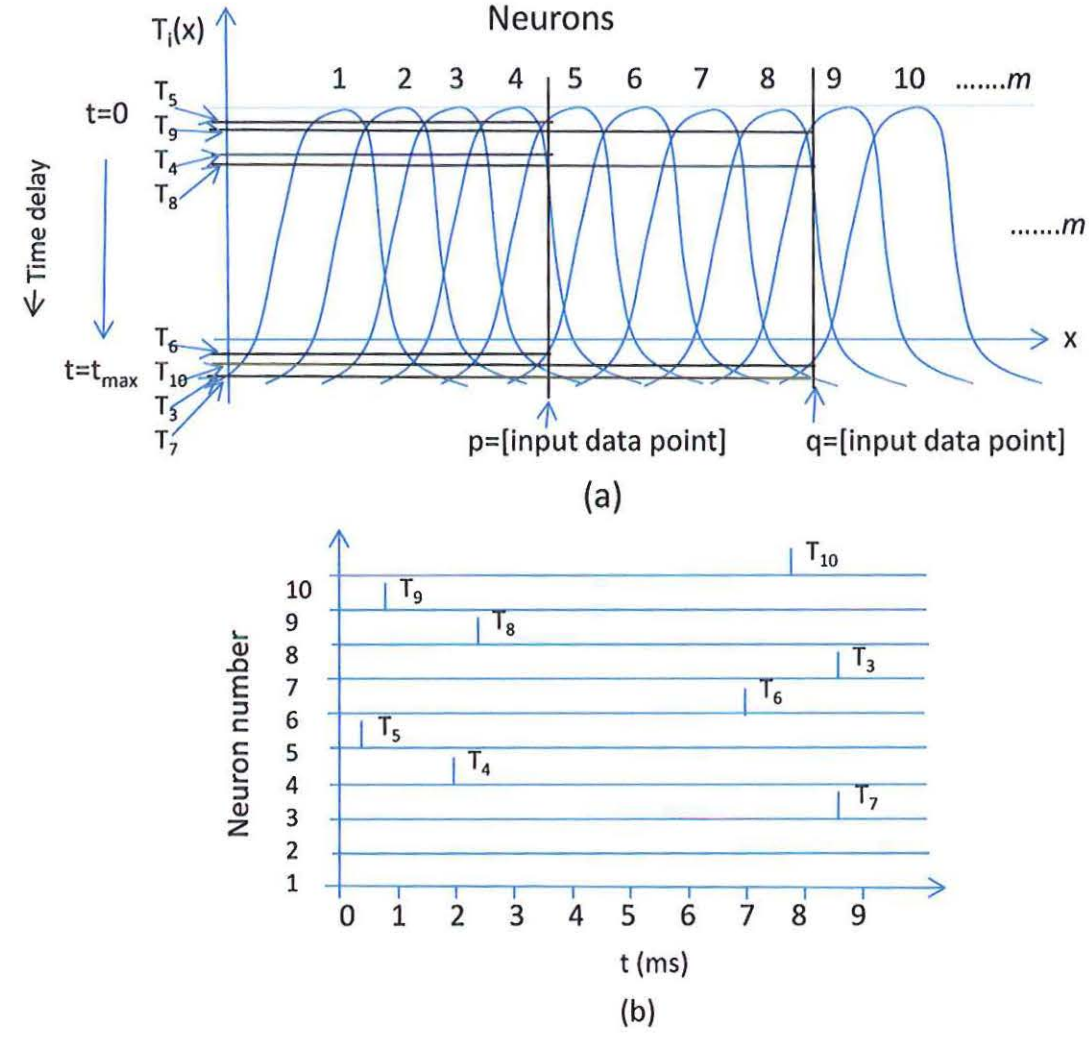


Figure 3.3. GRF encoding scheme. (a) 10 sensory neurons distributed evenly across input space, with data points *p* and *q* as inputs. $T_i(x)$ are the corresponding time-since-input at which the neurons will spike [30]. (b) Spike timings for *p* and *q* when encoded.

Figure 3.4 illustrates the firing patterns of a sequence of values [1, 4.4, 9.2, 2, 6.4, 10, 3, 8, 7, 5] input one after another with a 10 ms delay between each input. With no overlap, these are easily distinguishable by any heuristic examiner. However, Algorithm 3 is capable of encoding inputs which are not forced to wait for the prior input's spike pattern to be fully generated before beginning to generate its own, which results in the same input stream consisting of overlapping spikes, as shown in Figure 3.5. This arrangement of overlapping values would be very difficult to reverse the encoding on with no prior knowledge of to which original input a given spike in an overlapping frame applies.

Algorithm 3 Encoding using overlapping GRFs [30]

```
1: Initialize number of neurons $m$ and coefficient of multiplication $\gamma$ used in (18).
2: Initialize the range of inputs, $n_{min}$ and $n_{max}$, to the encoder.
3: for each neuron $i$ do
4:     Compute centers of each Gaussian functions, $C_i$ using (17)
5: end for
6: Compute the width of Gaussian functions, $w$ using (18)
7: for each neuron $i$ do
8:     for each sample input $x$ do
9:         Compute the magnitude of firing, $f_i(x)$, using (19)
10:        Calculate the time delay values $T_i$ from $f_i(x)$ using (20)
11:        Set neurons with $T$>9ms to not fire
12:    end for
13: end for
```

Algorithm 4 is designed to disentangle these with no prior knowledge of the original inputs other than the delay time between inputs. It accomplishes this by passing the spike timings Δt within τ of a given input's start time t through (21) to get a hypothetical activation value $f_i'(x)$ that the spike would have if it were caused by the input entered at time t. This is then filtered through (22) to determine what the hypothetical value x' would have to be to generate f_i'.

$$f_i'(x) = \frac{1-\Delta t}{\tau} \qquad i = 1,2,\ldots,m \tag{21}$$

$$x' = \sqrt{\left(-2\times w^2 \times \frac{\log\left(f_i'(x)\right)}{A} + C_i\right)} \qquad i = 1,2,\ldots,m \tag{22}$$

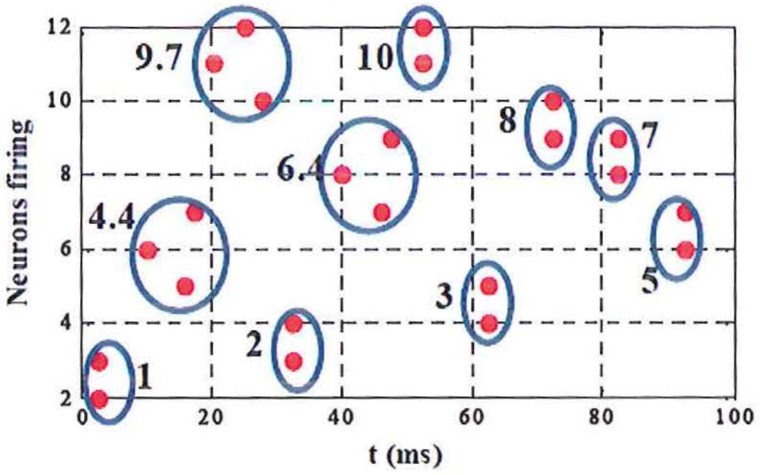


Figure 3.4. Spiking response for input sequence [1, 4.4, 9.7, 2, 6.4, 10, 3, 8, 7, 5] with a 10 ms pause between inputs. Since no input can have a spike more than 9 ms after it is entered, there is no overlap [30].

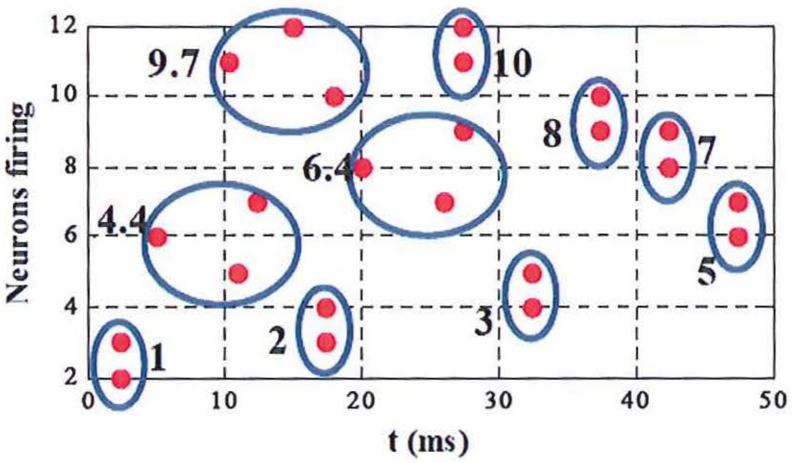


Figure 3.5. Spiking response for input sequence [1, 4.4, 9.7, 2, 6.4, 10, 3, 8, 7, 5] with a 5 ms pause between inputs. Overlap occurs between the last 5 ms of potential response time of one input and the first 5 ms of potential response time of the next [30].

There will be two to three spikes which truly belong to the input given at time *t*. These spikes will all have the same *x'* value. All other spikes, caused by other inputs, will generate different *x'* values which will not match. Taking the statistical mode of all hypothetical values *x'* will thus return the true value *x* which was input at time *t*. As can be seen clearly in Figure 3.6, the GRF encoding scheme can successfully encode the sine wave given in (16) into spikes from which the original sine wave can be reconstructed according to Algorithm 4. With arbitrary temporal resolution, the recovery is precise and accurate to arbitrary degrees.

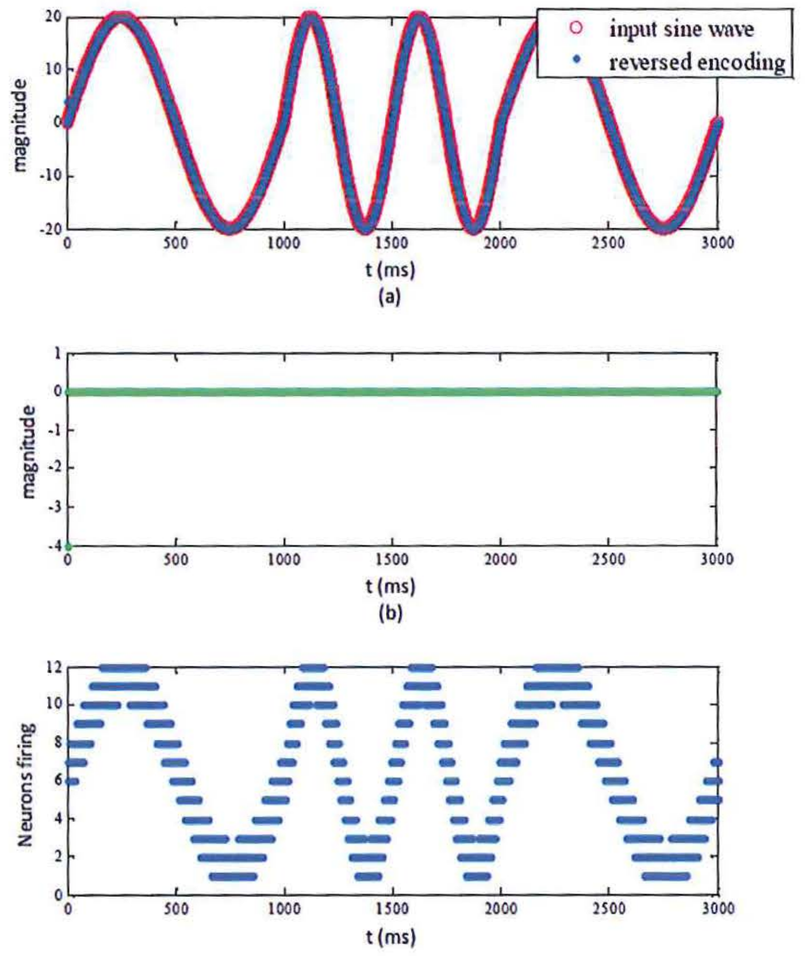


Figure 3.6. GRF reversibility demonstration. (a) Original sine wave overlaid by the sine wave recovered from the encoded spikes. (b) The error is shown to be non-existent; GRF loses no information with arbitrary temporal resolution. (c) Raster plot of the overlapping inputs encoded as spikes [30].

Algorithm 4 Recovering data from GRF-created spikes [30]

```
1: Initialize $t_{threshold}$ as the maximum possible delay of a
   spike after its triggering input
2: for each sample time $t_0$ do
3:     for each spike between $t_0$ and $t_0 + t_{threshold}$ do
4:         Calculate $\Delta t$ = (time of spike) – ($t_0$)
5:         Use $\Delta t$ in (21) to determine $f'(x)$ for every spike
6:         Use $f'(x)$ in (22) to calculate $x'$ for both the
           positive and negative roots
7:     end for
8:     Take the statistical mode $x$ of all spikes' $x'$ for
       every sample which gives the recovered input
9: end for
```

It is perhaps unsurprising that the coded spike train resembles, cosmetically, the original uncoded signal, when one considers that the sensory neurons' Gaussian receptors are positioned along the signal's domain.

3.3.3. Dual-Neuron *n*-bit Representation (DNNR). Developed in-house in [29], this encoding method inputs an entire continuous value into an SNN in a single time step (usually 1 ms). A particularly simple two-step process translates the continuous value on a prescribed range first into an *n*-bit Grey code and then into a pattern of *n* spikes arrayed across 2*n* sensory neurons. Two neurons represent any one bit in order to distinguish between binary input values of "0" or "1" and no input at all, as demonstrated in Figure 3.7. When no input is present, none of the sensory neurons are firing. Each neuron pair has one designated as "odd" and the other as "even." When an input is present, exactly one of the two neurons in each pair fires. "Odd" neurons firing indicate the bit represented by that particular pair is "1," while "even" neurons firing indicate the bit represented by that particular pair is "0."

The range and resolution of this encoding method (as shown in Algorithm 5) is strictly determined by the number of bits (and thus the number of sensory neurons) used to encode the values. In the work done in [29] and [30], 12 bits and therefore 24 sensory neurons were used, giving a maximum of 4096 discrete values possible. For a range of slightly more than [-20, 20], this gives two decimal places of precision. That's very nearly continuous resolution from a human observer's standpoint.

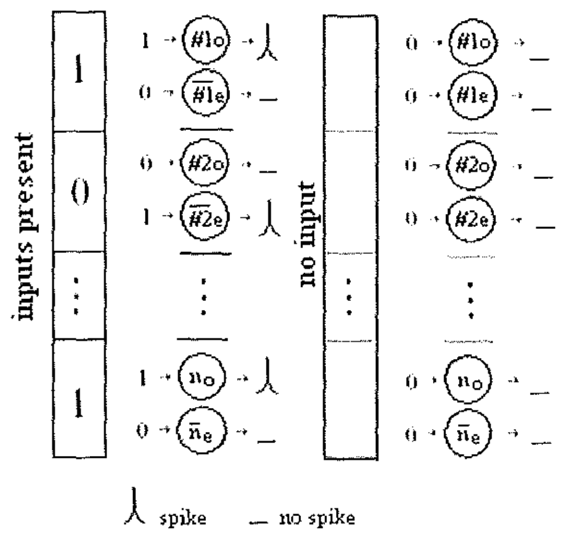


Figure 3.7. Encoding method for n bits with two neurons, and odd (o) and an even (e), each. A Grey code is used to transform real-world values into a binary representation; each bit is represented by two neurons [29].

Algorithm 5 Encoding a real-world value via the dual-neuron n-bit representation spiking encoding method [30]

```
1: Based on resolution and range of real-world values, initialize the number of bits $n$
2: Create $m$ spiking input neurons, where $m = 2n$
3: while there are inputs to pass to the sensory organ:
4:     Convert current input to $n$-bit Grey code
5:     Pass the bits of the Grey-coded value to odd-    numbered neurons
6:     Pass the complement of the same bits to the even-numbered neurons
7:     Neurons (even or odd) which receive a "1" output a spike, neurons which receive    a "0" do not output a spike
8: end while
```

Because there is no time-dependence on the information, this is a strictly spatial encoding method. All of the information encoded by Algorithm 5 is passed through the sensory neurons at once. This very straight-forward encoding method also means it is easy to tell that the only place information is lost is in the resolution: in the experiment shown in Figure 3.8, any decimal value less than 10^{-2} is truncated to the second decimal place. Algorithm 6 outlines the reversing of the encoding based on the spike stream, which is simply the same sinusoid on which the GRF encoding method was tested.

Algorithm 6 Reversing the DNNR spiking encoding method to recover the original input [30]

```
1: Initialize $n$ to half the number of neurons used in the encoding
2: while there are inputs to read from the sensory organ:
3:         for each pair of neurons
4:         if the odd neuron is spiking and
                   the even neuron is not
5:              Record corresponding bit as a "1"
6:         else if the odd neuron is not spiking and
                   the even neuron is
7:              Record the corresponding bit as a "0"
8:         else
9:              There is no input here, break to next input
10:        end if
12:        Convert $n$ Grey-coded bits to real-world value
13:        Move to the next input
14:    end while
```

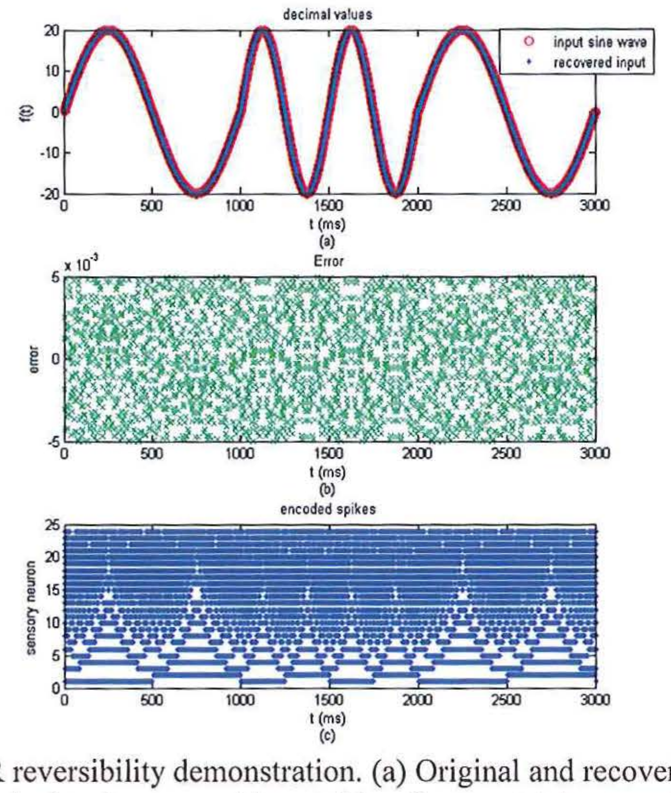


Figure 3.8. DNNR reversibility demonstration. (a) Original and recovered input; (b) error between the original and recovered input; (c) spikes containing encoded values [30].

The regular, well-organized spikes encoding the data are easy to read. With 12 spikes in every input, the energy fed to the SNN by the sensory neurons is very constant, as well. Reversibility is easily demonstrated because the spikes correspond directly to a Grey code, which is 1:1 related to the original continuous input value. The error falls neatly in the range of less than 0.01, precisely as predicted for the resolution of this encoding method.

3.3.4. Scaling the Inputs. Because GRF and DNNR both have finite ranges over which they can encode continuous values (and varying resolutions for said ranges), it is important to be certain the encoder is constructed and calibrated to handle both the range and resolution desired. The experiments demonstrated here used a range of values from -20 to 20 for the input streams. DNNR utilized 12-bit Grey code with a resolution of two decimal places. Twelve-bit Grey code can achieve 4096 unique values, which enabled a two-decimal-place resolution to encompass slightly more than the range of -20 to 20, but it was the minimum number of bits to achieve that range and resolution.

It is possible to increase the range of the DNNR by one of two methods: either scale the Grey code such that its 4096 unique values correspond to a broader range (which comes at the expense of resolution falling to less than two decimal places), or add more bits. Both of these are relatively simple, requiring either a scaling constant before the Grey encoding or a slight modification to the Grey encoder to account for having more bits available. The DNNR encoding method uses a linear two neurons per bit, so scaling in this fashion is likewise linear.

Scaling GRF input is not much more difficult. The number of neurons corresponds directly to the number of Gaussians available. The more tightly-packed the Gaussians are and the finer the temporal resolution used, the higher the resolution of the continuous valued numbers possible. In the experiments presented here, the temporal resolution is 1 ms. The centers C_i of the Gaussians are computed for the range of values desired and the number of available neurons according to (17). Increasing the scale simply requires re-calculating C_i for the new range. Adding more neurons will increase resolution by providing more spikes per input. Increasing the temporal resolution improves resolution by simply making for finer distinctions between spike timings.

PREM scales more easily, as simply entering the values you desire as the rate $\lambda_{a,b}$. It requires that the lower bound of possible inputs be determined based on the number of potential paths being used, so that an arithmetic adjustment can be made to ensure all input values will be greater than this lower bound, but it otherwise is as easily scaled as any second-generation NN's inputs.

3.4. COMPUTATIONAL ENGINE: POLYCHRONOUS SPIKING NETWORK

Having discussed three possible means of encoding continuous values into spikes, it now behooves us to examine exactly what it is those spikes will be input into. The Izhikevich model of spiking neurons is discussed in Section 2.2 [40], and it was briefly mentioned that they are arranged into a polychronous spiking network [43], but what exactly does that mean?

The PSN is a network of N Izhikevich neurons, 80% of which are excitatory with parameters set to "regular spiking" according to Figure 2.3, and 20% of which are inhibitory and set to "fast spiking" according to the same figure. The network is sparsely connected (10% connectivity) by assigning each neuron $N/10$ output synapses. Each synapse has a weight, a delay, and a destination neuron. When an excitatory neuron spikes, it sends that spike to each synapse, and the voltage strength of that spike is multiplied by the weight. A number of time steps k (set to k=1 ms in Izhikevich's model) after the spike equal to the delay associated with the synapse in question, the destination neuron sees the transmitted and weighted spike, which modifies its $v(k)$ through the I variable. Inhibitory neurons work identically, save their contribution to I in their destination neurons is negative, tending thus to reduce the likelihood of a neuron spiking.

Highly recurrent, sparsely connected reservoirs of neurons – spiking or otherwise – are not a new addition. ESNs and LSMs have used spiking neurons and second-generation neurons for quite some time. The innovative addition to Izhikevich's PSN is, rather, the delays, which lead to polychronous clustering. Without these variable-length delays, every neuron connected to a firing neuron by a synapse will see the incident spike at the same time. Spatial calculations are the only things possible with this arrangement, as what neuron is connected to what is all that matters.

Still, the connection weights are important! They are trained according to a method known as spike timing-dependent plasticity (STDP). This unsupervised training mechanism determines which presynaptic neurons are "interesting" to the postsynaptic neuron, and weakens the "uninteresting" ones while strengthening the "interesting" ones. This is accomplished by observing the postsynaptic neuron's responses. Each time the postsynaptic neuron fires, incident spikes' timings from each of the presynaptic neurons are checked in a short time before and after the postsynaptic firing. Presynaptic firings which are incident upon the postsynaptic neuron prior to the firing of the postsynaptic neuron are deemed "interesting," because they contributed to the firing of the postsynaptic neuron. Those which arrive "late" – that is, after the postsynaptic neuron fires – are deemed "uninteresting," and are weakened. The closer in time to the postsynaptic firing that the incident spikes are, the more the associated synaptic weight is strengthened or weakened. Figure 3.9 illustrates the STDP curve.

Equation (23), as shown in Figure 3.9, gives the time-dependent weight adjustment equations [46]. When a presynaptic neuron arrives prior to a post-synaptic spike firing, the synapse is *potentiated*, or strengthened. When a post-synaptic spike is fired before the arrival of a pre-synaptic spike, that synapse is *depressed*, or made weaker.

$$\Delta s = \begin{cases} A_+ e^{-\frac{\Delta t}{\tau_+}} & \Delta t = t_{post} - t_{pre} > 0 \\ A_- e^{\frac{\Delta t}{\tau_-}} & \Delta t = t_{post} - t_{pre} \leq 0 \end{cases} \tag{23}$$

The constants in (23) are the maximum depression A_-, the maximum potentiation A_+, and the temporal windows of interest τ_+ and τ_-. In [46], these are set to maximum weight adjustments of $A_{+/-}$=+/-0.004, pre-spike watch window τ_+=15 ms, and post-spike watch window τ_-=20 ms. These were also used in the unsupervised training of all Izhikevich BSNNs implemented for the experiments performed in this dissertation.

One unusual property of a BSNN is the need to pre-train it unsupervised on the kind of data on which it will be expected to operate. This process is known as "maturation," and is performed to let the STDP process optimize the weight strengths so that relevant spiking patterns emerge as natural responses to various inputs. This process is fairly straight-forward: the BSNN is exposed to inputs of the type and range on which it will be operating once mature, and the STDP process is allowed to run.

Typically, the expert assessment that a BSNN of Izhikevich neurons is matured after 6-12 hours of simulated time is assumed to be true, but scientifically, it would be nice to have a metric for measuring the maturation of a BSNN. Algorithm 7 outlines this relatively straight-forward procedure. Connection weights have a minimum of zero, but are arbitrarily assigned a maximum (often "ten," but it can be anything that is not "effectively infinite"). As STDP runs, the weights of neurons typically saturate at the minimum and maximum, based on the spike patterns most excited by the input stream. While they may shift in value, they typically do not stay in the middle of the possible ranges of weight values for long. The percentage of synapses whose weights fall within a chosen band of values near the minimum and maximum can be used as a metric to measure the maturity of a BSNN. Determining the size of the bands and the percentage of neurons that must be within them to count as "mature" is a matter that remains to be studied, and six hours is sufficient on BSNNs made of Izhikevich neurons.

For a 100-neuron PSN, these weights separate and stabilize by as early as 50 simulated seconds in, as shown in Figure 3.10. The separation is also visible in Figure 3.11, which shows the progression of each of the synaptic weights associated with the PSN's excitatory neurons. The training inputs are 225s of power system generator data of the sort used in Section 4.5. Examining a 1000-neuron PSN trained on the same data required increasing the number of synapses per neuron to 75 to ensure enough activity to keep the PSN spiking. This dramatically increases the number of synapses, but merely seems to smooth out the percentage-distribution plot given in Figure 3.12. However, the sheer number of synapses firing makes any separation far less obvious; Figure 3.13 shows synaptic weights covering the entire range.

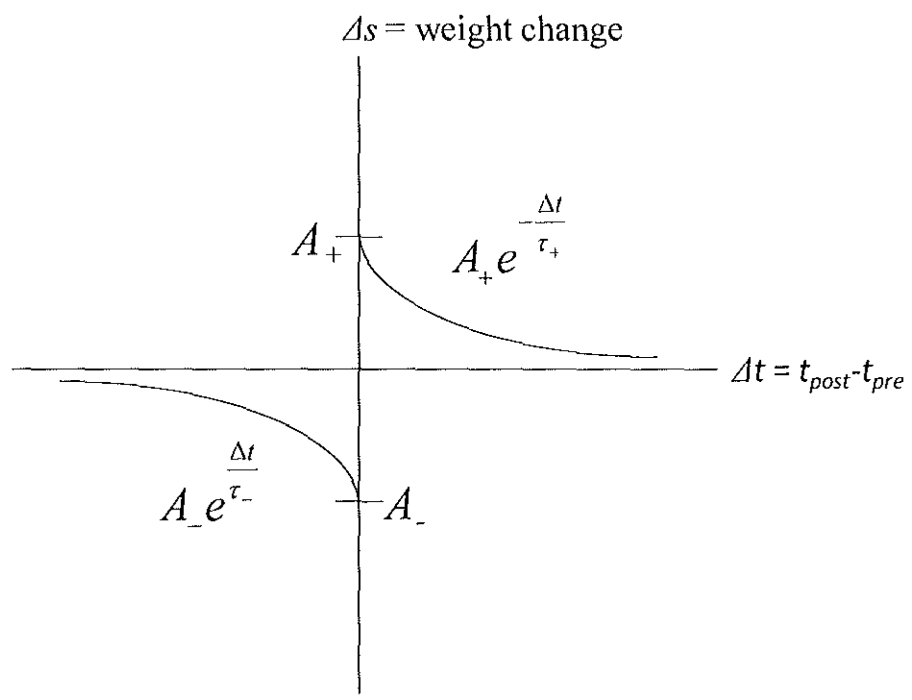


Figure 3.9. STDP weight adjustment curve. The weight s is adjusted by $\Delta s(\Delta t)$. The time between firings Δt is the time of the postsynaptic neuron's firing minus the time of the presynaptic neuron's firing [46].

If all we have are connection strengths – synaptic weights – it becomes crucial to consider that the spikes fired at one time instant will arrive at all destination neurons simultaneously. A fully-connected BSNN with this model relies strongly on some weights being too weak to cause an incoming spike to regularly have an impact on the spiking of the postsynaptic neuron, lest continuous bursting across all neurons be the sole result. This can be avoided with sparser connectivity, but still limits the spike patterns to simultaneous spikes on multiple neurons to create spiking responses in postsynaptic neurons.

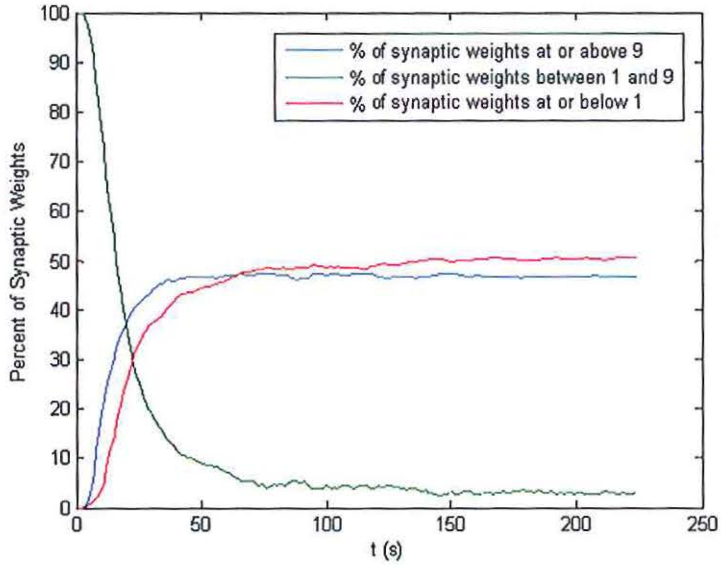


Figure 3.10. Percentage of excitatory synaptic weights in a 100-neuron PSN with 80 excitatory neurons equal to or greater than nine, less than or equal to one, and in between one and nine, as a function of maturation time. Notably, though the number above nine has plateaued by 50s, the number at or below one is still slowly rising even by 225s.

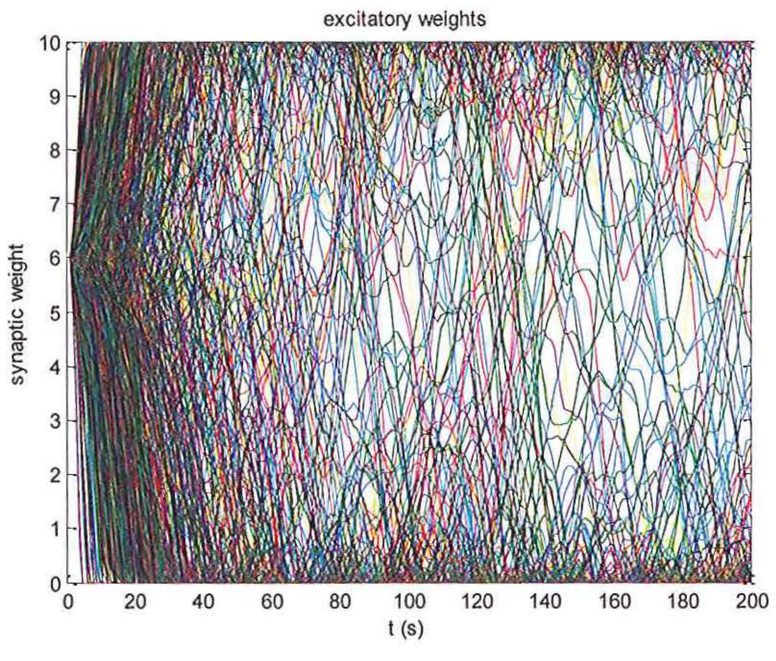


Figure 3.11. Change over time in excitatory weights of the 100 neuron PSN shown in Figure 3.10 during maturation.

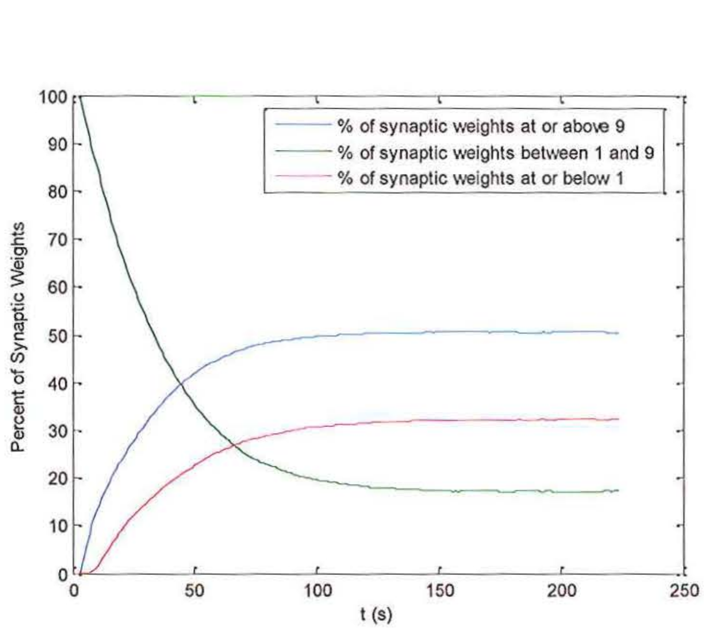


Figure 3.12. Distribution of synaptic weights over 225 simulated seconds of maturation via STDP. Smoother than Figure 3.10, it still is slowly increasing though nearing a plateau by 200s.

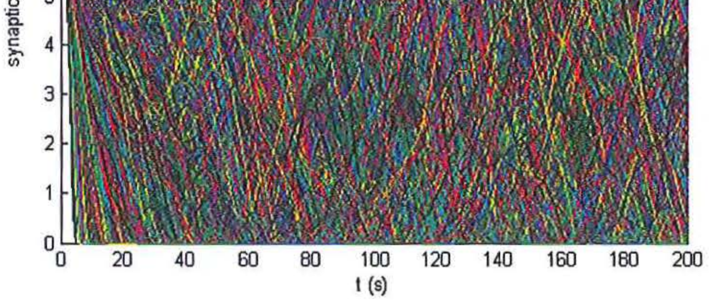


Figure 3.13. Excitatory weights of a 1000 neuron PSN matured over the course of 200 simulated seconds. Spread is easy to see, but the constant jumping from top to bottom causes this to have less obvious a separation of instantaneous values than in Figure 3.11.

Algorithm 7 Maturing a PSN

```
1: Initialize a PSN of $N$ Izhikevich neurons with 10% connectivity, synaptic
   weights $s$ with strengths randomly distributed between 0 and 10, and
   delays between 1 and 10 ms.
2: for seconds from 1 to 3600*6 (six simulated hours)
3:     for milliseconds from 1 to 1000 (one simulated second)
4:         simulate the Izhikevich neurons for one millisecond
           according to (10) and (11)
5:         record each neuron that fires in this millisecond
6:         increase weight change $\Delta s$ which connect to a
           neuron that fired in previous milliseconds according
           to (23)
7:         decrease weight change $\Delta s$ which neurons that fired
           this millisecond would stimulate if the post-synaptic
           neuron fired in prior milliseconds according to (23)
8:     end for
9:     add accumulated weight changes $\Delta s$ to weights $s$
10: end for
```

An alternative is to introduce variable conduction delays. This introduces a property known as *polychrony*, which refers to the capability of different temporal patterns of spikes across multiple neurons to trigger specific groups of neurons in response as the delays line up properly. These *polychronous groups* or clusters represent specific responses of a polychronous BSNN, or polychronous spiking network (PSN), to specific stimuli. The varying delays enable the same sets of neurons firing to trigger wildly different polychronous clusters depending on the order in which they fire, a property denied to BSNNs with a single fixed delay for all synapses. A given neuron's firing is thus an identifier for specific patterns in space and time through the network as a whole.

Izhikevich demonstrates a five-neuron polychronous network with delays chosen to create 14 polychronous groups, two of which were self-propagating cycles, in [43]. There are already more polychronous groups than there are neurons, and there are 20 synapses, so the number of groups is approaching the number of synapses even in this small network. Finding connection delays using this same method while adding an additional neuron leads to the connection diagram with delays as shown in Figure 3.14.

This arrangement produces 37 polychronous groups, shown in Figure 3.15. A fully-connected six-neuron network only has 30 synapses. Even at six neurons, a polychronous network can contain more distinct polychronous groups than there are connections between neurons!

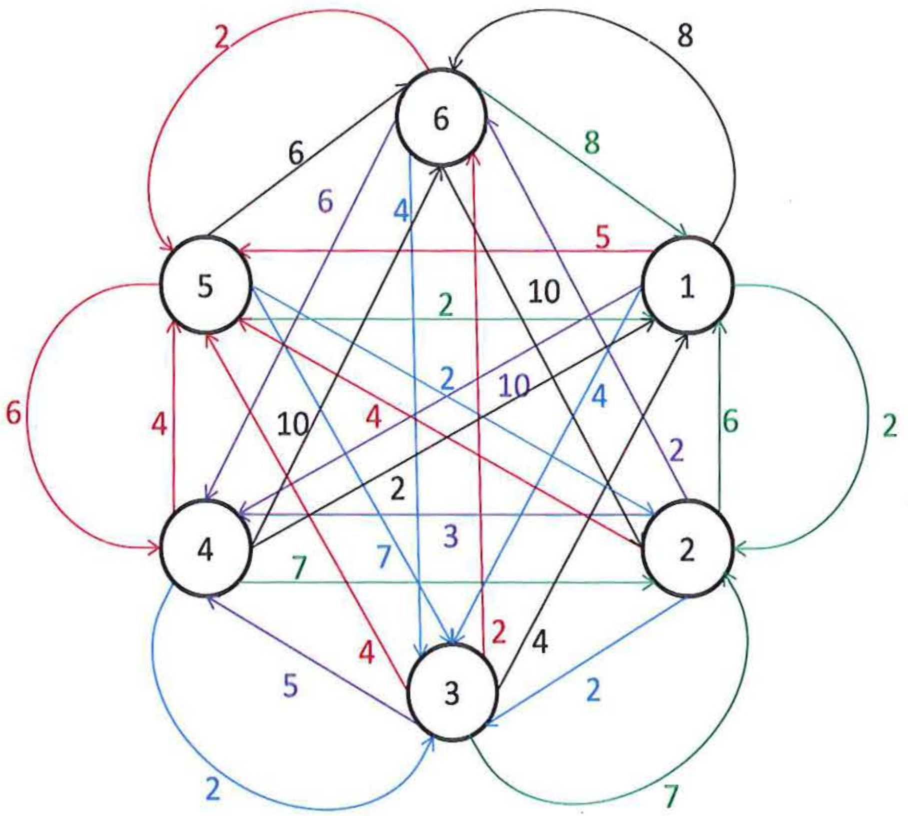


Figure 3.14. Connection diagram with delays for a six-neuron network that produces the 37 polychronous groups shown in Figure 3.15. Thirty-seven polychronous groups produced by the weights shown in Figure 3.14.

Each group can be identified to a specific pattern, if the problem is the sort of pattern-matching and categorization that has been done over the last decade. But with each additional neuron exponentially increasing the number of polychronous groups, the number of patterns possible swiftly exceeds the threshold necessary for high-resolution pseudo-continuous calculations of the sort modern computers and second-generation neural networks already perform. It's easy to believe that, at such a rate of growth in numbers of polychronous groups compared to addition of neurons, biological brains with tens of thousands to hundreds of *billions* of neurons might have more polychronous groups than there are particles in the universe. The challenge then becomes translating the groups' responses into meaningful numeric outputs in a continuous regime.

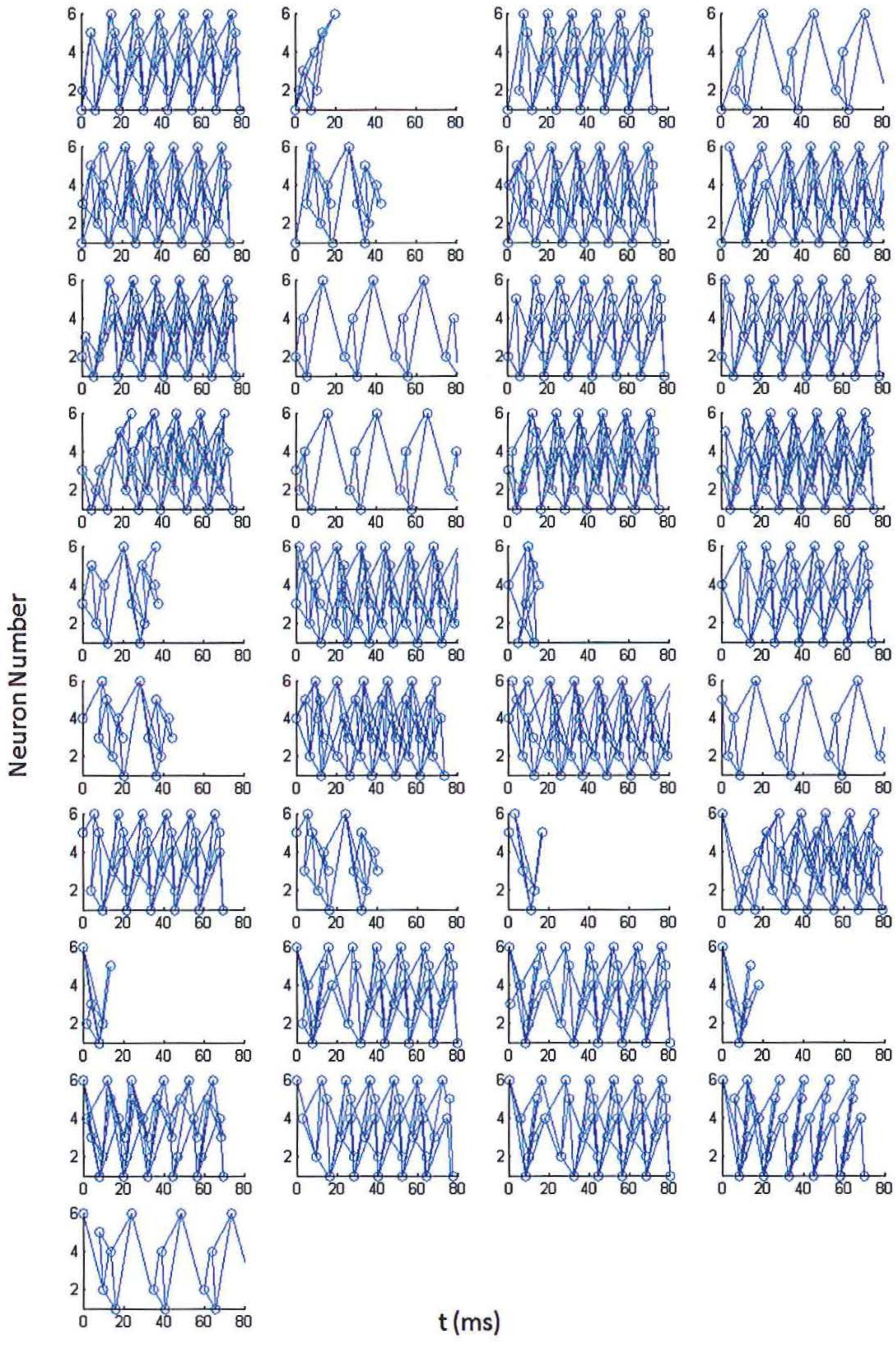


Figure 3.15. Thirty-seven polychronous groups produced by the weights shown in Figure 3.14.

3.5. DECODING

With a PSN that has literally astronomical calculation capacity and a means of inputting arbitrary continuous values into it with confidence that the values survive the encoding process, the final step is finding a way to extract the desired dynamics from the PSN and translate them into continuous valued numbers with meaning in the real world. There are a couple choices in how to map the PSN's responses to real-world values. Izhikevich has a Matlab script that will build a PSN and find polychronous groups; it may be possible to hand-map various groups to various outputs by determining which respond most strongly to given inputs, and form a functional mapping that way. This, however, seems very time consuming and inefficient; a look-up table would probably be a better choice. In fact, as Figure 3.16 illustrates, such a mapping would effectively *be* a look-up table.

The other method is to find a way to translate the spikes coming out of the PSN neurons into real-world values and tune them to the target with a decoder. The dynamics of the PSN contain the information about the inputs, and with so rich a response structure, it is likely that any desired functional relations can be isolated with proper tuning of the decoder. The PSN does the heavy lifting of creating the dynamical responses; the tuner simply has to isolate which ones are desired. Such a method is described here.

One of the big contributions of this work is the development of such a decoder, which finds the temporal responses of the PSN neurons and translates them into continuous values by using Gaussian receptors spread over a moving watch window of the last 20 outputs from a given neuron. The Gaussian functions are optimized for maximum response over 1000 simulated seconds in order to determine what delay and distribution of spikes represents the strongest responses from the neuron.

The timing of the spikes relative to the beginning of the watch window is used as input to the Gaussian receptor field, and then all responses are summed to generate a numeric output that represents the strength of response of the neuron, as shown in Figure 3.17. The mechanism for optimizing these Gaussians is a form of unsupervised training using Particle Swarm Optimization (PSO) [47]. A training signal is given to the PSN in a series of iterations of the PSO.

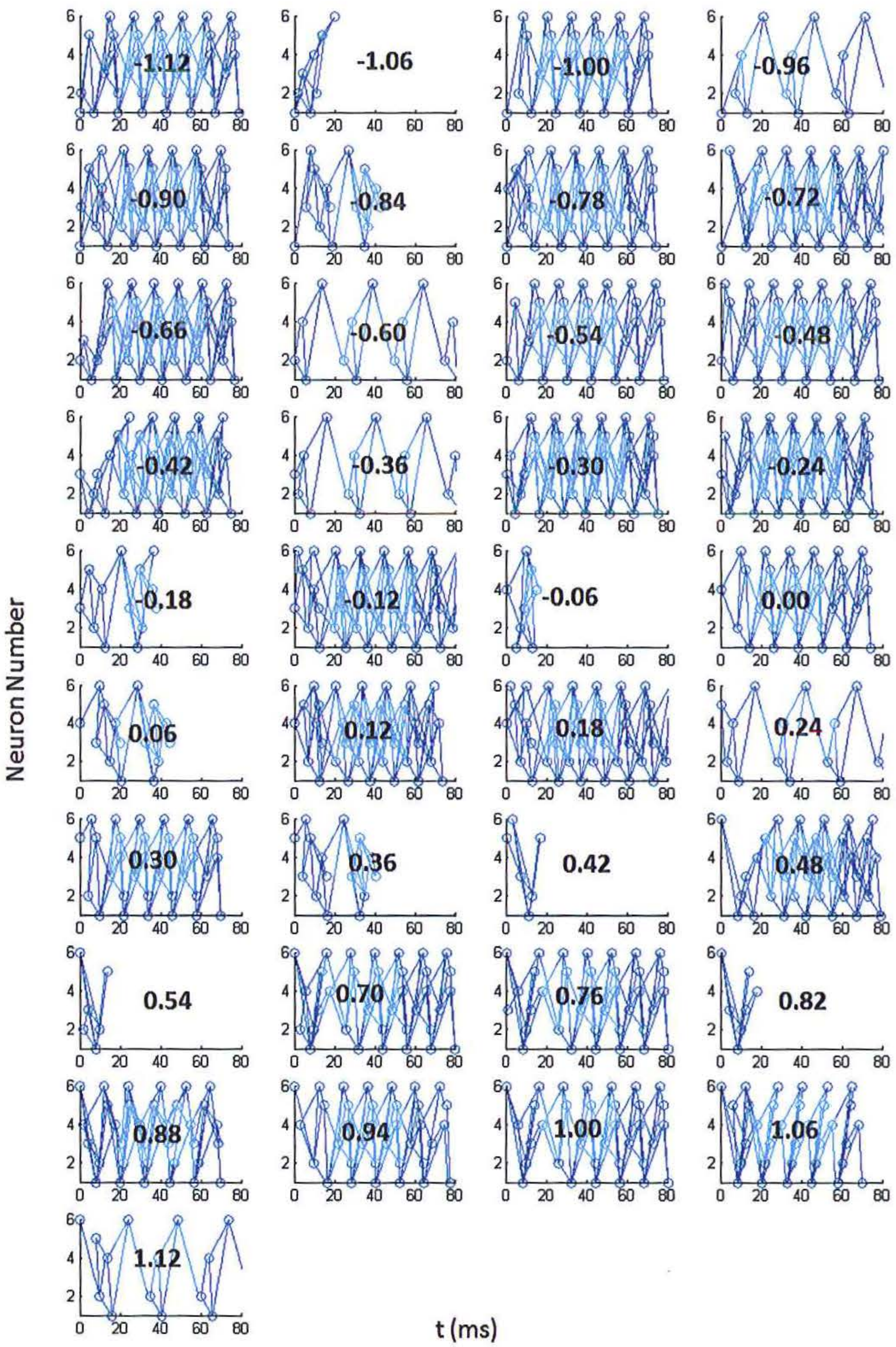


Figure 3.16. A possible mapping of the polychronous groups identified in the six-neuron PSN shown in Figure 3.14, enabling the PSN to map from -1.12 to 1.12 in increments of 0.06.

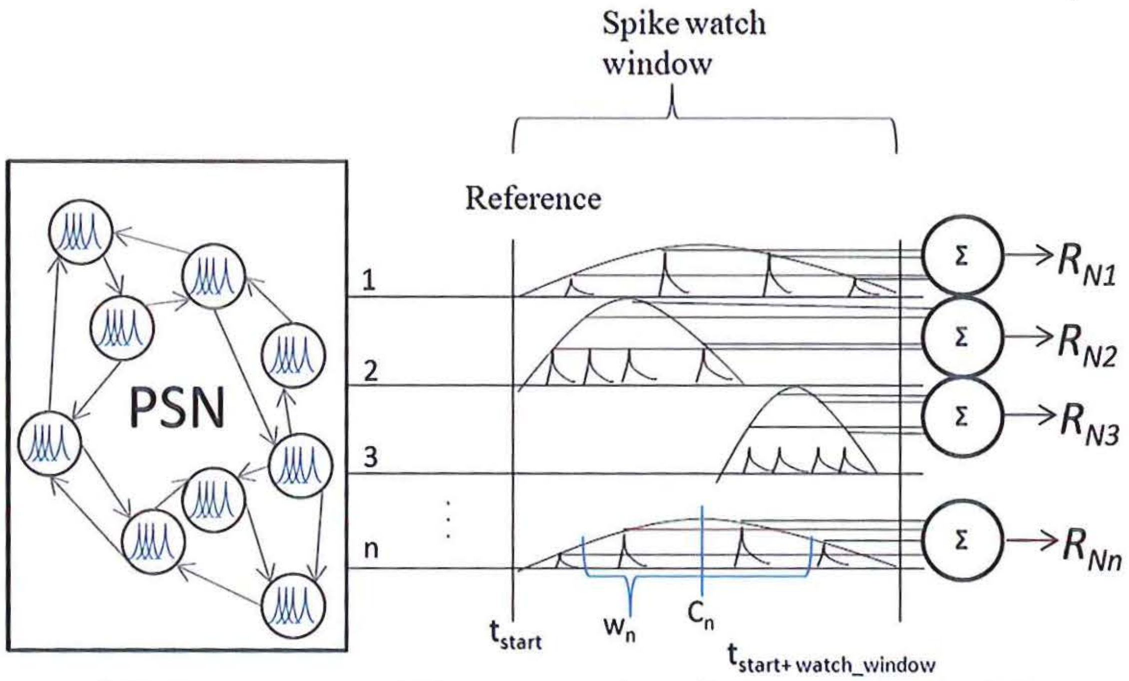


Figure 3.17. Gaussian temporal filters transforming spike trains in a watch window into a real-world numeric output. The center C_n and width w_n are shown for the n^{th} Gaussian function. A PSO is used to optimized C_i and w_i for each neuron *i*. R_{Ni} are continuous-valued real numbers from each neuron *i*.

The signal should be a sample of the sort of data on which the trained PSN and its decoder will be expected to operate. The sine wave in (16) is used to train and test the decoder in this Section. The signal for the real-world problem of power system identification would require a sample power system signal. Each iteration sums the output *R* for neuron *i* of the watch-window filtered through the candidate Gaussian function. The objective function (24) seeks to maximize this sum.

$$fitness_i(k) = \sum_{j=k-watch\_window}^{t} R_i(j) \tag{24}$$

Each neuron *i* has its own Gaussian function, which needs to be optimized independently of the other neurons'. Each neuron therefore has its own independent PSO. Each neuron in the PSN has *m* (typically 20 or 30) particles of two dimensions each. These dimensions are the width w_i and center C_i of the Gaussian assigned to the neuron in question; the Gaussian function is normalized so that the amplitude is inversely proportional to the width in order to ensure there are two competing goals and "maximum width" is not the pure optimum.

The center of the Gaussian picks a length of time after a given input corresponding to the watch window. Spikes close to this time-delay will cause a stronger response through the Gaussian. It takes fewer spikes in this time frame than it does earlier and later in the watch window to generate a strong response. The width of the Gaussian is inversely tied to its amplitude by the requirement that the Gaussian be normalized. A wider Gaussian will respond strongly to spikes that appear far from its center as well as near; a narrow Gaussian will respond much more strongly to spikes closely clustered around its center and will nearly ignore spikes well away from it. Figure 3.18 illustrates some centers and widths found for a 100-neuron PSN used in case studies in this Section.

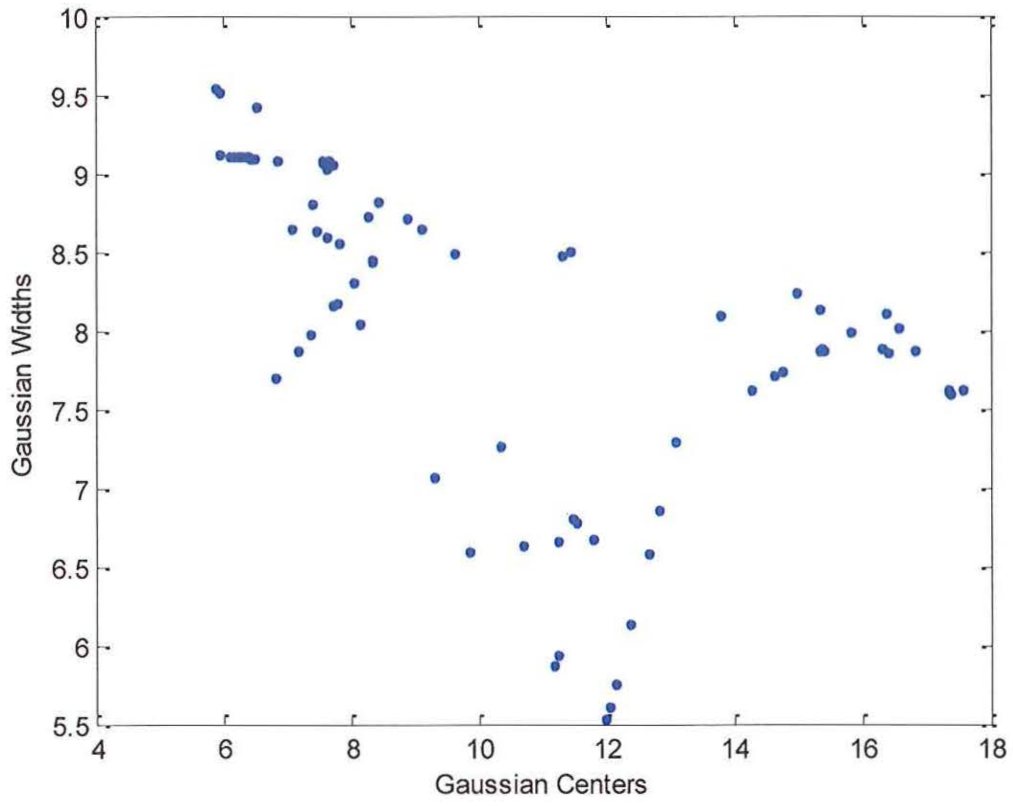


Figure 3.18. Centers and widths of the Gaussian functions trained for the 100 neuron PSN used for the case studies in this Section.

The timing of spikes from a neuron can contain information in a couple of ways. Sudden clusters of spikes grouped close together in time convey information in their bursting presence, indicating an event to which they correspond. The rate at which spikes fire can, alternatively, convey information in their rate, be it some sort of clock-like timing or a representation of some numeric significance. Optimizing for the highest possible sum of responses, the PSO will find the centers and widths such that neurons prone more to bursts of spikes close together that represent specific event-responses will have those groups identified clearly, while those whose information is mostly based on overall rate of fire will tend to have Gaussian functions which are spread out and simply get their value from the overall rate at which the neuron fires during the watch window.

Because a PSN theoretically needs to mature for at least 6 simulated hours (12 is better) [43], these Gaussians were trained at the same time in the experiments presented here. Figure 3.19 provides a flow chart explaining the procedure. The maturing PSN's weight adaptation due to STDP, however, means that a lot of the calculation cycles spent on the PSO pre-maturation were likely wasted as the dynamics of the neurons changed with each iteration. It is faster to do this in two stages: first, mature the PSN, then run the PSO for 20-100 iterations to optimize its widths and centers, thus not wasting any of the cycles. The PSO typically plateaus before 10 iterations have passed, as shown in Figure 3.20.

The fact that it finds an optimum and does not improve thereafter so quickly implies that the centers and widths are relatively insensitive, but that there is some optimality to be found. Because it requires so few PSO iterations to find a reasonable optimum, performing this in a second step as suggested previously saves a great many computational cycles compared to running the PSO algorithm concurrently with the PSN maturation.

By finding the shape and relative center of a Gaussian accumulator for spikes across a watch window in a given neuron which maximizes that neuron's numeric response representation, the PSO finds the Gaussian which best captures the interesting dynamics of that neuron. When spike responses relevant to polychronous clusters stimulated by the input signal to which the PSO is sensitizing the decoder occur, the trained Gaussian will respond with high values. When "noise" occurs, the trained

Gaussian will tend to have weak responses. This allows an overall strength of the decoder's numeric response to indicate how strongly the polychronous clusters of which this neuron is a part are firing. The remainder of the decoder disambiguates these response strengths by picking which ones are relevant to the desired target output.

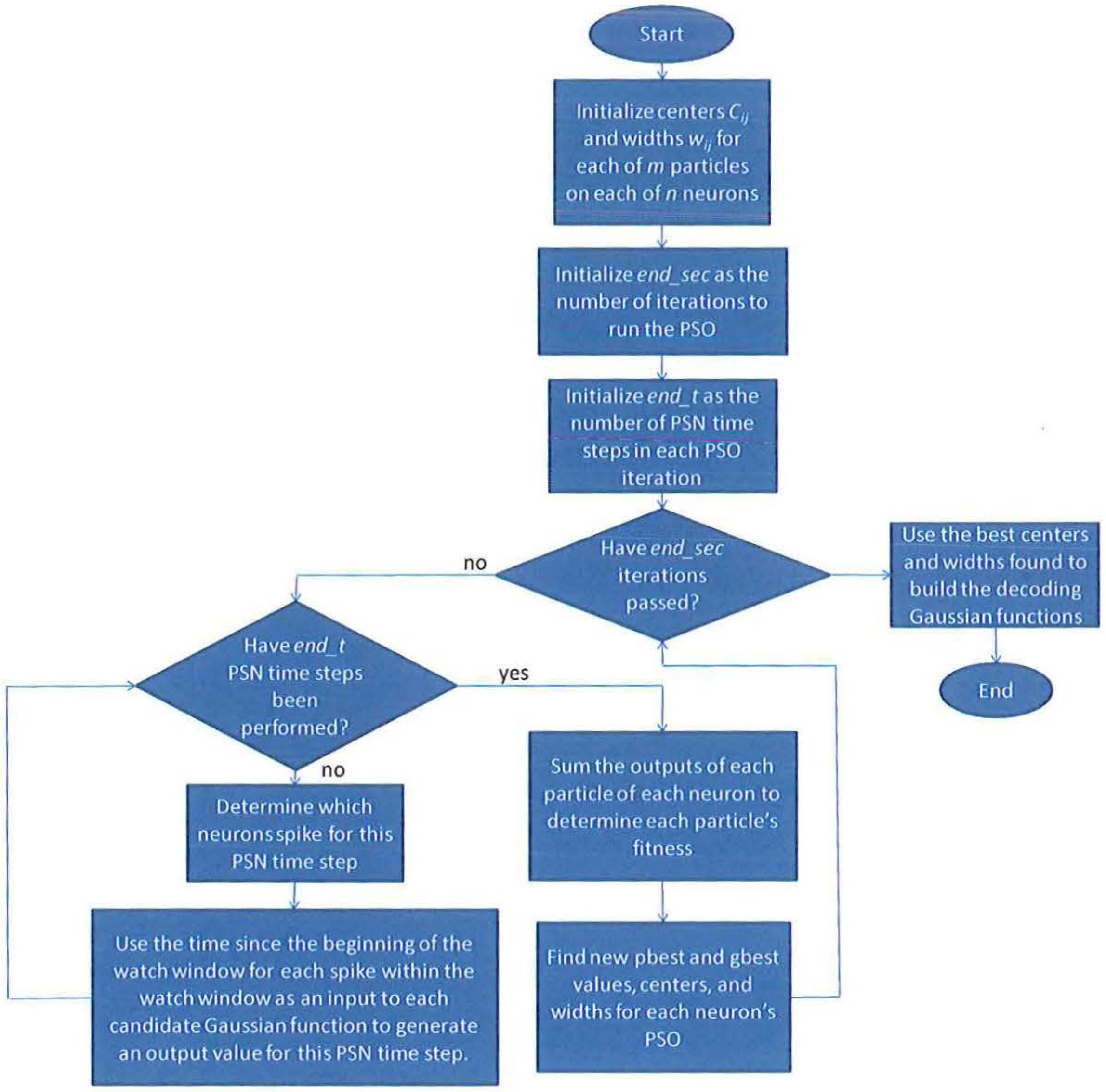

Start
Initialize centers C_{ij} and widths w_{ij} for each of m particles on each of n neurons
Initialize *end_sec* as the number of iterations to run the PSO
Initialize *end_t* as the number of PSN time steps in each PSO iteration
Have *end_sec* iterations passed?
no
Use the best centers and widths found to build the decoding Gaussian functions
End
Have *end_t* PSN time steps been performed?
yes
no
Determine which neurons spike for this PSN time step
Use the time since the beginning of the watch window for each spike within the watch window as an input to each candidate Gaussian function to generate an output value for this PSN time step.
Sum the outputs of each particle of each neuron to determine each particle's fitness
Find new pbest and gbest values, centers, and widths for each neuron's PSO

Figure 3.19. Flow chart showing the procedure for using a PSO to determine the centers C_i and widths w_i for each neuron.

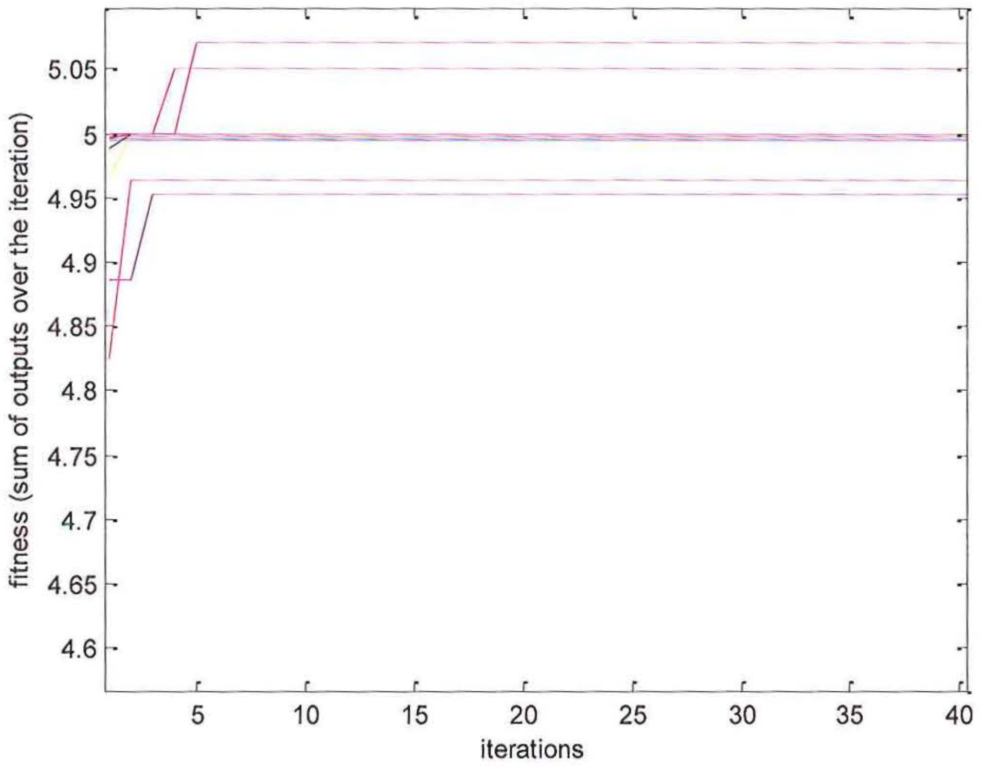


Figure 3.20. Fitness of selected Gaussian functions as decoder filters for the watch windows of their respective neurons. The PSO plateaus before 10 iterations are passed; this seems to be typical behavior.

The decoding Gaussians provide *N* outputs every time step *k*. The outputs are all continuous-valued numbers that probably do not have a whole lot of easily-discernable meaning to the naked eye. However, they represent the information contained in the spike patterns of each of the *N* neurons to which they are respectively associated. A first-order attempt at tuning these outputs to find the important dynamics that contain the information present in the PSN's responses is to use a simple weighted sum, as shown in Figure 3.21.

The weights are trained with gradient descent to find the relative importance of each PSN neuron's output to the desired dynamics, and the weighted sum is thus tuned to produce the desired target.

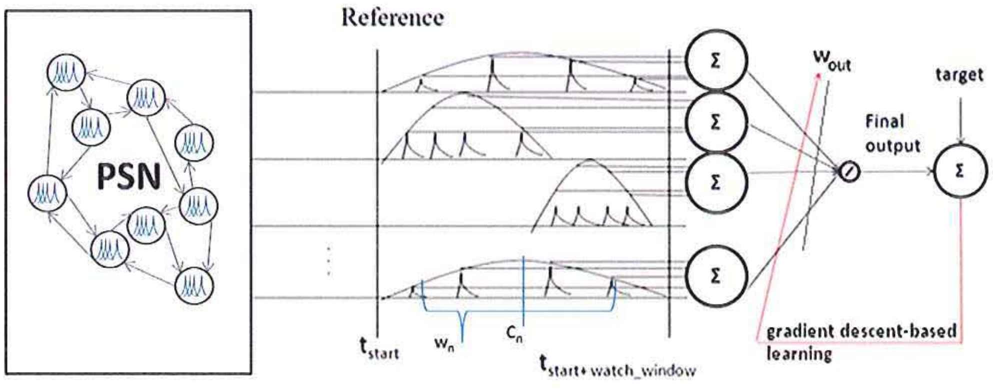


Figure 3.21. Simple sum output, trained with gradient descent error.

This first-order sorting of the neuron outputs enables important neurons to be emphasized and less important ones (with respect to the desired dynamics) to be de-emphasized, but may not be able to sort fine dynamics from the mix. Figure 3.22 shows a more elaborate solution: construct an MLP of second-generation sigmoidal neurons with the outputs from the PSN as the input layer and with one hidden layer before the final output layer.

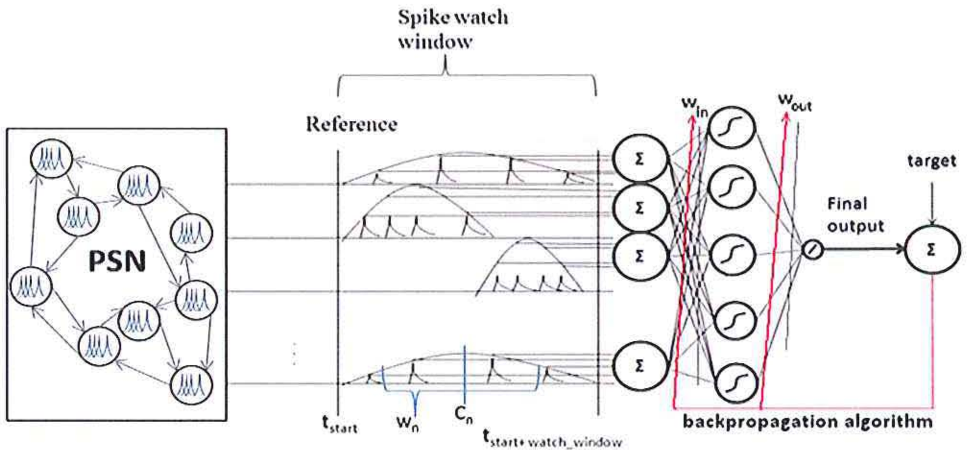


Figure 3.22. MLP output tuning, trained with backpropagation.

STDP remains on in the PSN, allowing the PSN to adapt to changes in inputs and have diverse and relevant dynamics over time. The training of the tuner's weights – whether an MLP or a simple weighted sum – is also online. It remains a matter of some research whether to use a moving window batch training for this "online" mechanism or to continue with output-by-output gradient-descent learning. The latter is demonstrated on two applications (function approximation and time series reproduction) in the next Section.

3.6. CASE STUDIES

3.6.1. Time-Series Reproduction. Completing the experiment begun in Section 3.3, (16) is passed through a GRF encoder and fed to the PSN with the intent to train the decoder to reproduce the original signal. This is done on a 100-neuron PSN with 12 sensory neurons. In the raster plot in Figure 3.23-Figure 3.25 a), neurons 1-80 are excitatory, 81-100 are inhibitory, and 101-112 are sensory. A very close examination of the raster plot can reveal a hint of the sinusoid generated by (16) in the spiking cascades of the excitatory neurons.

Figure 3.23-Figure 3.25 all follow the same format: b) is the very first iteration of exposure to the input signal, c) is the penultimate iteration, and d) is the final one. Gradient-descent-based training (including backpropagation where appropriate) is performed online for all but the last iteration; the last iteration is done with the weights of the decoder fixed to test whether it has learned. Figure 3.23 is generated using the simple weighted sum decoder in Figure 3.21, while Figure 3.24 and Figure 3.25 use an MLP decoder. Figure 3.24 has five sigmoidal neurons in the hidden layer, while Figure 3.25 has 10.

It is fairly easy to see the improvement between Figure 3.23 b) and Figure 3.23 c), as the initial iteration is fairly clearly following a delayed, slavish adjustment likely due strictly to the weights being forced to match what the target signal called for one or more time steps too late. In c) and d), however, though the signal is fuzzy and far from perfect, it's clear that its timing is dead-on with the target signal. The simple output weights have learned to find the dynamics, if not to suppress all the noise. Figure 3.23 d) is the most encouraging, though, because there is no weight adjustment. The weights there are fixed.

The responses of the decoder to the PSN are not being dynamically adjusted to follow the target; they have truly learned the target. This not only demonstrates that the decoder can find the target (albeit not with the level of precision one might desire), but that the target information must actually be present in the PSN's own responses.

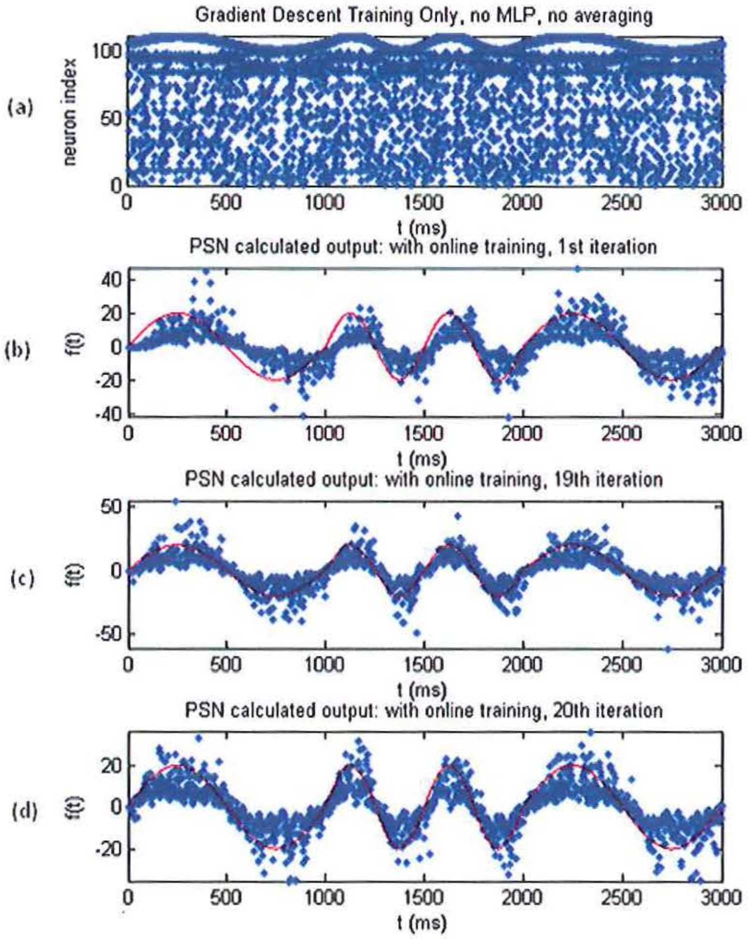


Figure 3.23. Gradient-descent trained weighted sum of the PSN output. a) Raster plot of the last iteration. b) First iteration of exposure, online gradient-descent training. c) Penultimate iteration of exposure, online gradient-descent training still going on. d) Final iteration, no training of decoder weights.

Replacing the simple weighted sum tuner with a five-hidden-neuron MLP results in Figure 3.24. Interestingly, Figure 3.24 b) is the "prettiest" following of the target signal, and it's the very first iteration. This is absolutely because the backpropagation is forcing the weights to slavishly follow the target. The fuzzier response in c) is actually encouraging because it means the MLP has its own ideas of what the result should be, and the weights are adjusting less powerfully. That it has learned something of the signal is clear in d), when, as in Figure 3.23, the decoder weights are fixed and the target is never confirmed to the MLP. However, it clearly hasn't quite learned all the dynamics of even this simple system; the testing run in d) does not follow the target above 10 or below -10.

Increasing the number of hidden neurons to ten, however, as in Figure 3.25, enables the MLP-based decoder to learn the signal quite well. Notice how the MLP-based decoders lack the noise and uncertainty of the simple weighted sum-based one. They are still fuzzy, but crisper by far than Figure 3.23's efforts. And the testing run in plot d) in all three of them seems to indicate that the decoder has learned to find the desired target dynamics! However, it is possible that the PSN is actually not doing anything but providing noisy stimulus to the decoder, and the decoder's own second-generation neural components are memorizing a pattern to be repeated no matter the input.

Such a charge would seem impossible, since no recurrent nor time-delay elements exist within the weighted summation device nor the simple MLP, and thus no memory of what states it has already output can be present, but it is still worth proving that the PSN output truly is a relevant input to the decoder, and that the decoder is not making up target data out of whole cloth. Figure 3.26 shows similar data to Figure 3.23, but only for 20 iterations. Figure 3.26 a) shows the sinusoid from (16) as an input, with the final plot in each demonstrating a testing run with the decoder's weights fixed. Figure 3.26 b) shows the same network, trained the same way and with the decoder trained by the same target, but with a completely random input to the sensory neurons.

It can clearly be seen in Figure 3.26 b) that the online gradient-descent training of the weights forces the weighted sum output to follow the target, albeit a few time-steps late. However, where Figure 3.26 a) has a testing run that looks about as good as its final training iteration, the testing run in Figure 3.26 b) is entirely noise. A close examination of the values of the y-axis even hints that the weights were forced to produce a narrow band based on any input at all. This narrow band remained constant once the weight training was turned off, so matches the final output band from the last online training iteration.

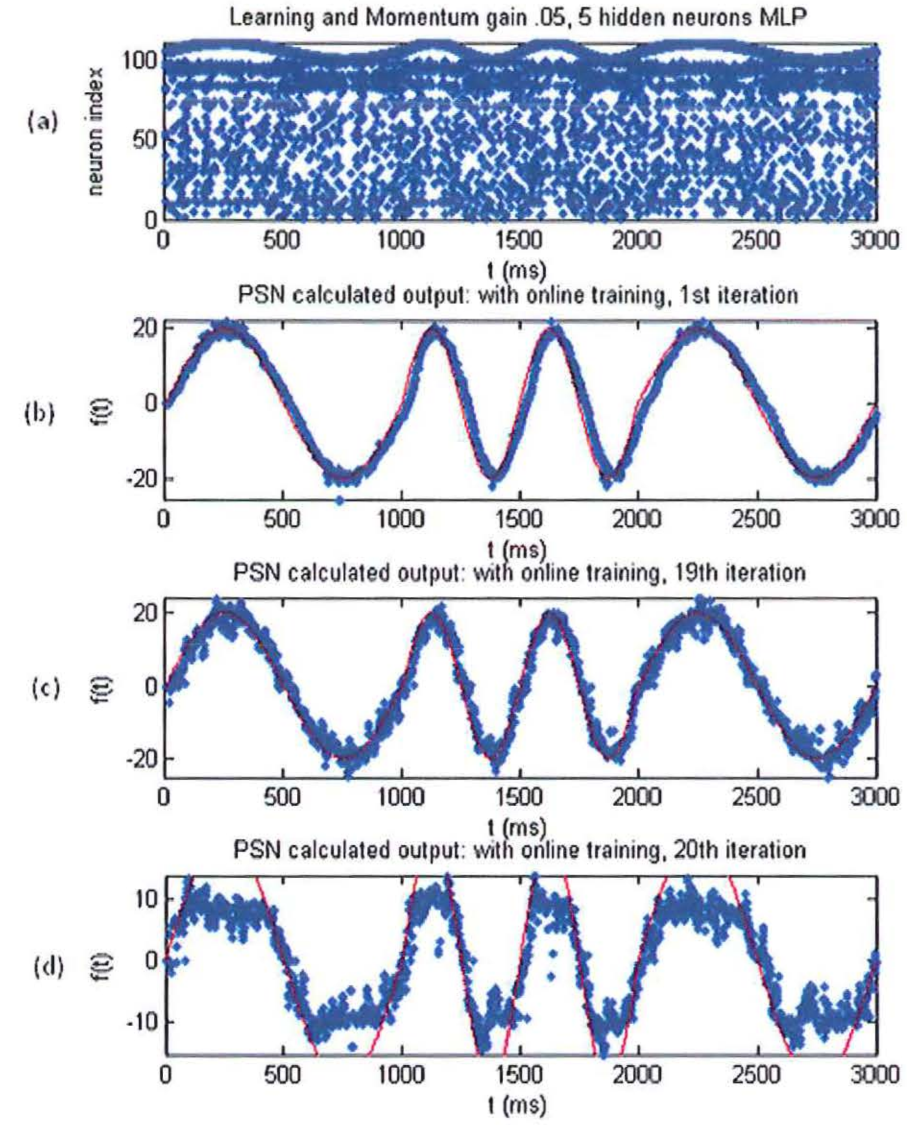


Figure 3.24. MLP decoding PSN output with five sigmoidal neurons in the hidden layer. a) Raster plot of the last iteration. b) First iteration of exposure, online backpropagation training. c) Penultimate iteration of exposure, online backpropagation training still going on. d) Final iteration, no training of decoder weights.

This is an encouraging result: for the first time, a continuous value set of inputs have been presented to an SNN, and the SNN's dynamics have been decoded into continuous values that are meaningfully related to the original values, reproducing with notable (if not perfect) fidelity the continuously-valued time series.

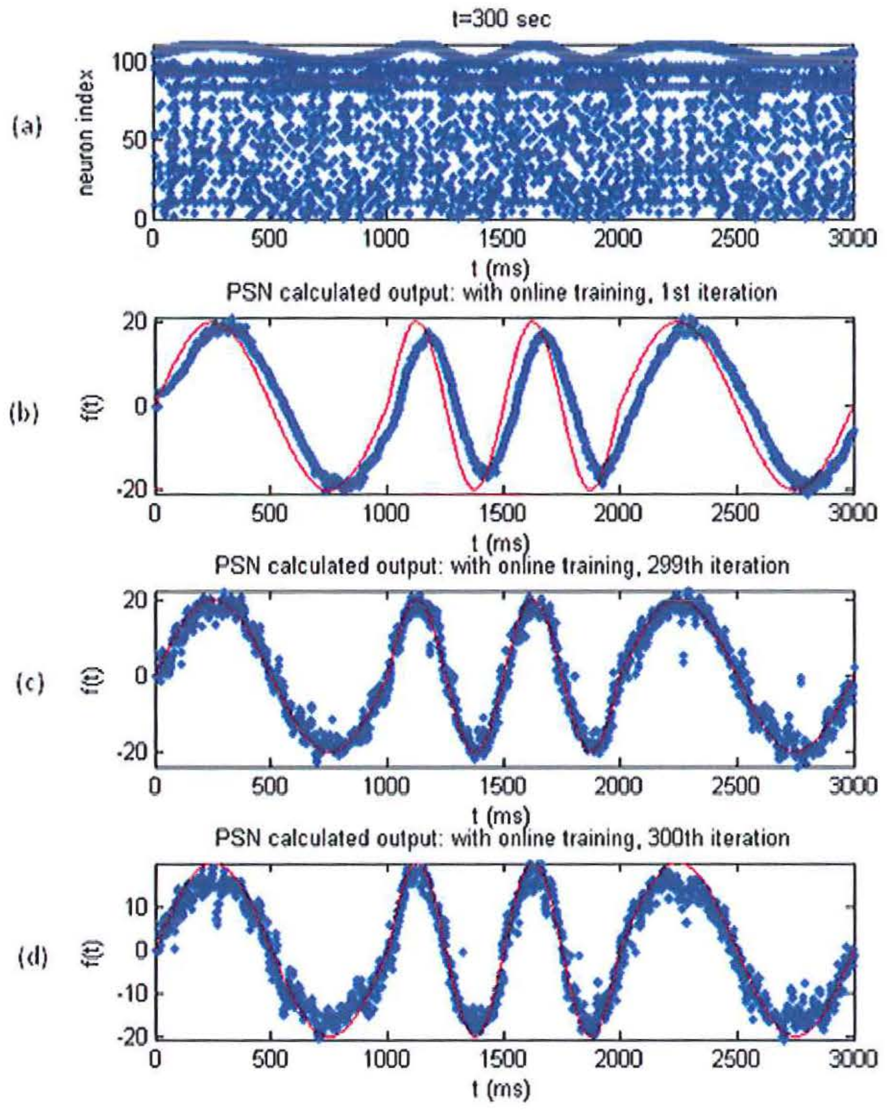


Figure 3.25. MLP output with ten sigmoidal neurons in the hidden layer. a) Raster plot of the last iteration. b) First iteration of exposure, online backpropagation training. c) Penultimate iteration of exposure, online backpropagation training still going on. d) Final iteration, no training of decoder weights.

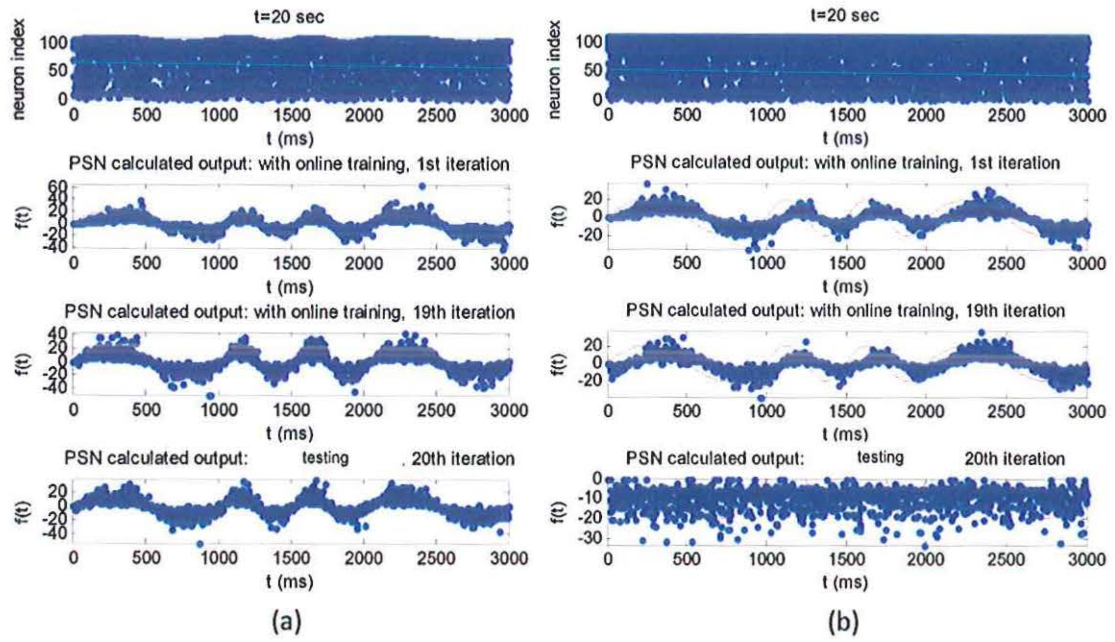


Figure 3.26. Comparison of training the decoder on a PSN fed by (16), and of training the decoder on a PSN fed by random spikes. (a) 19 iterations trained with (16). (b) 19 iterations trained with random noise. This clearly demonstrates, in the final testing run for both cases, that the decoder is not simply making up target information from whole cloth; it requires the PSN to contain meaningful dynamics related to the target input.

3.6.2. Function Approximation. Function approximation is actually a bit trickier than time-series reproduction and identification when using the PSN engine, because the PSN is a highly recurrent network with strong memory. Therefore, inputting a sequence of numbers with dependence on the prior input is in line with how the PSN's evolving dynamic states work, while inputting numbers with no dependence nor relation to their predecessor inputs leads to noise rather than constructive reinforcement of the state.

A simple sine wave is tested with GRF input (also temporal as well as spatial) that overlaps completely, one input per simulated millisecond. Figure 3.27 and Figure 3.28 look good while the online training is on, but the steady noisy state during the testing run reveals that they were slaved by the weight adjustment, not actually learning any dynamics which may not even be present.

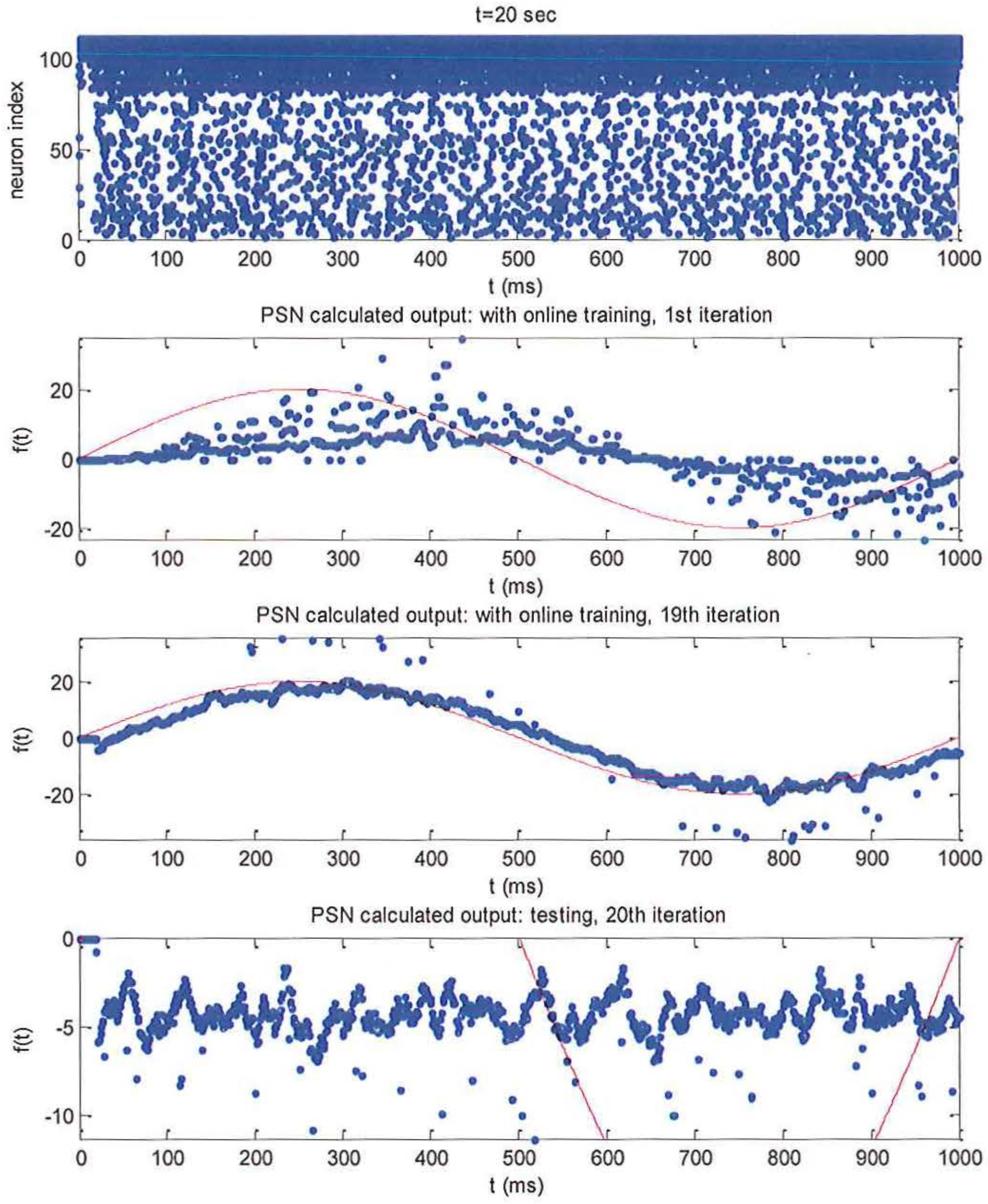


Figure 3.27. Weighted sum output attempting to learn to approximate a sine wave encoded using the GRF method. Iterations 1-19 used online gradient descent training on the output weights; iteration 20 keeps the weights fixed, and reveals a complete lack of learning.

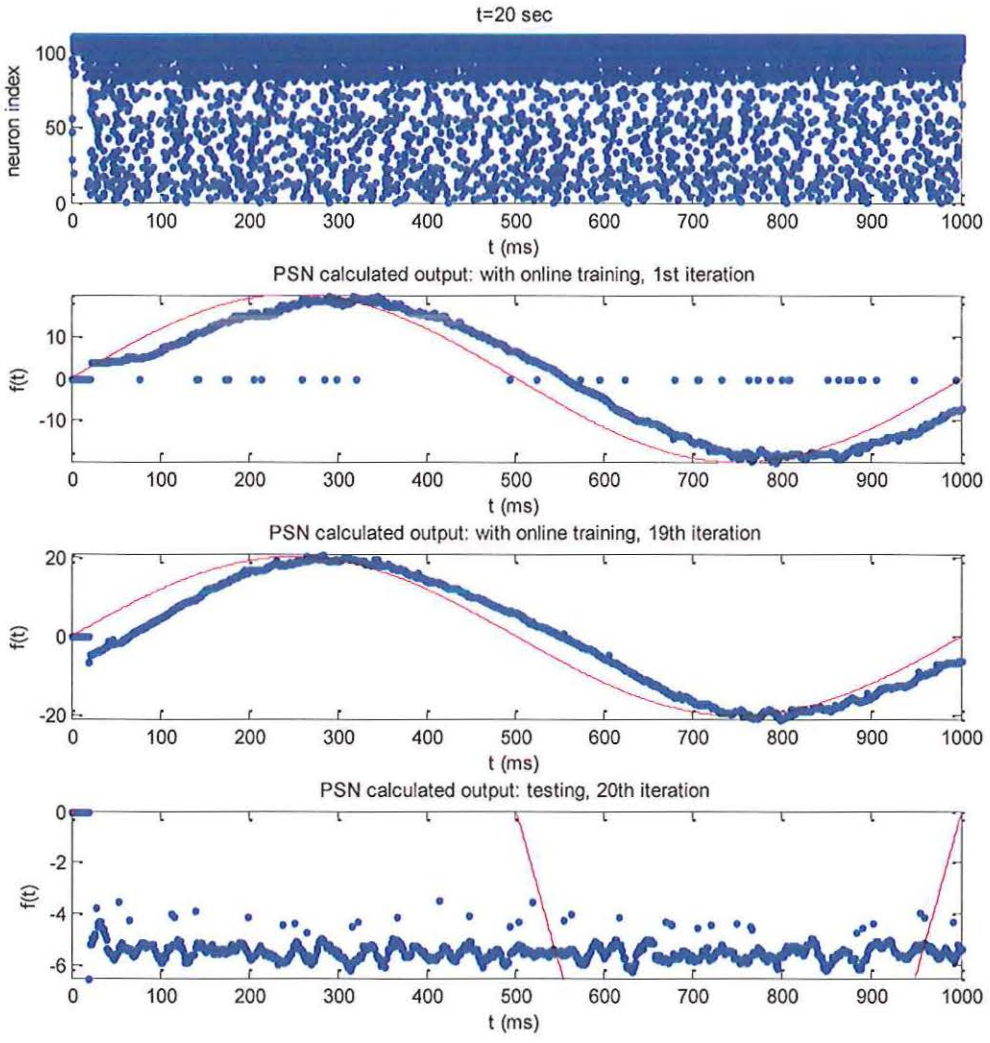


Figure 3.28. MLP decoder attempting to learn function approximation of a sine wave encoded using the GRF method. Iterations 1-19 used online backpropagation training on the MLP weights; iteration 20 keeps the weights fixed, and reveals a complete lack of learning.

3.7. SUMMARY

In order to develop BSNNs as viable successors to second-generation neural networks, the research presented here has demonstrated several methods for encoding arbitrary continuous-valued data into spike trains, input those into a PSN, and read out the dynamics of the PSN through a decoder tuned to isolate desired functional transformations to return arbitrary, continuous-valued outputs.

Three encoding methods – PREM, GRF, and DNNR – have been shown to transform numbers into spike trains and proven to contain the information so encoded. PREM is a purely temporal encoding scheme, but requires the largest number of neurons to encode data, as well as the most preprocessing. GRF is both temporal and spatial, and retains resolution down to a smaller number of neurons than either of the other two methods. For this reason, it is the preferred encoding method used in experiments in this dissertation. DNNR was developed in-house, and is purely spatial. Its greatest advantage is its capacity to input its entire value at once, but it tends to be less robust an input method than GRF.

The PSN is chosen as the computational engine because its rich dynamic response to the encoded inputs is based on the astronomical potential numbers of polychronous clusters, which hypothetically makes for one of the best-scaling computational architectures ever designed. The decoder developed uses a carefully-trained response function which isolates dynamics that indicate what a given PSN neuron finds "interesting," and relates it to the relevant dynamics. Weighted sum or MLP-based tuning then isolates the dynamics of interest to the problem at hand.

This section concludes with a case study of this BSNN framework on a time-series reproduction, and the problems of a time-independent function approximation with GRF encoding. This is somewhat unsurprising given the high temporal-dynamic dependence of the PSN's states on prior states; approximating functions with no time dependence from one input state to the next will require reducing the number of dynamics to be analyzed to find the relevant functional outputs.

Overall, however, the ability to reproduce an arbitrary, continuous-valued time series is a tremendous step forward in BSNN research, demonstrating that these networks absolutely can be used on problems that require functional calculation.

4. NEUROIDENTIFICATION OF GENERATOR DYNAMICS IN A MULTI-MACHINE POWER SYSTEM

4.1. INTRODUCTION

Today's power systems consist of interconnected machines that must maintain synchrony in order to avoid brownouts and blackouts. Governors and voltage regulators are present on generators in power systems in order to increase or decrease output frequencies in response to changing loads and maintain respective voltages. This enables them to maintain synchrony as loads change and voltage fluctuates across the power system.

Online identification of generator speed and terminal voltage characteristics are essential for fast and accurate control of modern power systems. Purely reactionary changes to the frequency as it stands at a given time *t*, however, can lead to overcompensating and driving the oscillations ever more wild rather than the damping that is desired. In order to facilitate the regulators' efforts and minimize the oscillations during adjustment, it is best to have some foreknowledge of what the system will look like in the near future. This enables the controllers to preemptively adjust generator outputs so that they are already counter-acting incoming oscillations and fluctuations when they arrive.

Classical controllers use linearized models to predict behavior around some nominal operating point. Such models are extreme simplifications from the real world, wherein a continuously changing environment causes the generator's dynamics to change as well, which can render the approximations around the operating point completely invalid. Intelligent designs for automatic voltage regulators (AVRs) and power system stabilizers (PSSs) are called for. ANNs are very effective tools for designing these types of intelligent controllers. In order to take the correct control action in a dynamically changing environment, an ANN based controller needs a neuroidentifier, which provides an estimation of the speed and terminal voltage characteristics of a generator from past values of speed and terminal voltage. The method of neuroidentification is also very effective for wide area monitoring and control [48] and finding dynamic equivalents of large power systems [49],[50].

As different ANN architectures were studied and their performances examined [48], [49], [51], [52], it became clear that MLPs, RBFs, RNNs, and echo-state networks (ESNs) do not actually represent the structure and function of biological neurons. To capture the scaling power of biological NNs, something more is needed.

4.2. NEUROIDENTIFICATION OF GENERATOR DYNAMICS

The two variables of primary interest in generator system neuroidentification are speed deviation and voltage deviation. In each power generator, there is a baseline speed at which the generator spins and a baseline voltage it maintains as its output. As the loads increase and decrease demand on the generator, these values deviate from that baseline for particular generators.

Classical controllers for generators are generally based on linearized models that predict system behavior obtained around a nominal operating point. As the system moves away from that nominal point, these classical models become increasingly inaccurate. They are thus able to handle small deviations, but become less capable the more desperately they are needed. Environmental conditions can also absolutely change the actual operating point from its nominal basis. In these conditions, classical controllers such as AVRs and PSSs have degrading performance. Neural networks are a possible alternative, serving as intelligent controllers. Neuroidentifiers provide an estimation of the current and future states of speed and terminal voltage deviation in the generators using current and past input and output values.

Because ANNs approximate the function, they need not be precisely on-target at all times. Their learning and adaptability means that a sudden change in the physical system being modeled will shortly be represented in the ANN as it adjusts to minimize the errors that suddenly occur due to the discrepancy between reality and the ANN's model. MLPs, RBFs, CNNs, and ESNs have all been used in neuroidentification experiments with rather impressive success.

However, power systems have tremendous numbers of inputs and call for equally tremendous numbers of outputs. Second-generation NNs share a scaling problem: the more inputs and outputs a ANN needs to handle, the increasingly-more hidden neurons the ANN must have to maintain accuracy. The hidden neurons scale exponentially with the numbers of inputs and outputs. Four-machine, two-area systems have been simulated for wide-area monitoring experiments, and it was found that the computational power required for real-time neuroidentification of the entire system was huge. This makes real-time calculation of ANN mechanics on real-world power systems – such as New England's 36 bus system – intractable.

Living brains do not face this scaling problem. They are capable of enormous amounts of parallel calculation on very fast time scales that regulate highly complex systems both autonomically and deliberately. SNNs are designed with more closely modeling living brain behavior and functionality in mind, and thus theoretically can be more accurate on larger amounts of data with the same number of – if not fewer – computational resources. Before this can be tested, SNNs need to be shown to be able to handle the same kinds of problems *at all*.

In this section, the IEEE 10 generator, 39 bus power system illustrated in Figure 4.1 is used as a test bed. Generators G7 and G10 are connected as shown to Time-Delay ANNs (TDNNs) which perform the neuroidentification.

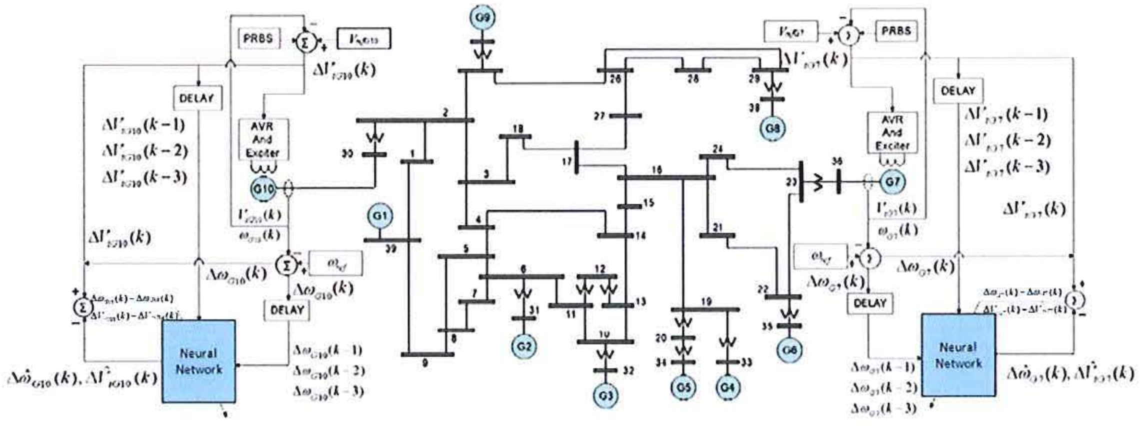


Figure 4.1. Schematic of the IEEE 10 generator 39 bus system with TDNN-based neuroidentifiers on generators G7 and G10.

The neuroidentification experiment is run twice, once by an ASNN, and once by an MLP using sigmoidal neurons. The multimachine power system shown in Figure 4.1 is simulated on the Real Time Digital Simulator (RTDS) in the Real-Time Power and Intelligent Systems (RTPIS) Laboratory at the Missouri University of Science and Technology. Both pseudorandom perturbations and simulated line-faults are performed, and the ASNN and the sigmoidal MLP are compared.

4.3. ASNN NEUROIDENTIFICATION

The parameters for the ASNN are given in Table 4.1. Most (α, ρ, τ, and r) are set according to values given in Section 2 and [25], while others are set by the experiment (20 hidden neurons) or through trial-and-error. They are largely unoptimized beyond some expert hand-tuning, as optimization of these parameters is beyond the scope of this dissertation. Figure 4.2 shows the ASNN implemented in a time-delay feedforward architecture.

Table 4.1: Parameters for the SNN neuroidentifier

SNN Parameter	Value
Number of inputs	9
Number of outputs	2
Number of hidden neurons	20
α	2
ρ	1
τ	Rand
r	1
Learning gain (η)	0.01
T_{ref}	0.5
V_{rest}	0
V_{thresh}	1

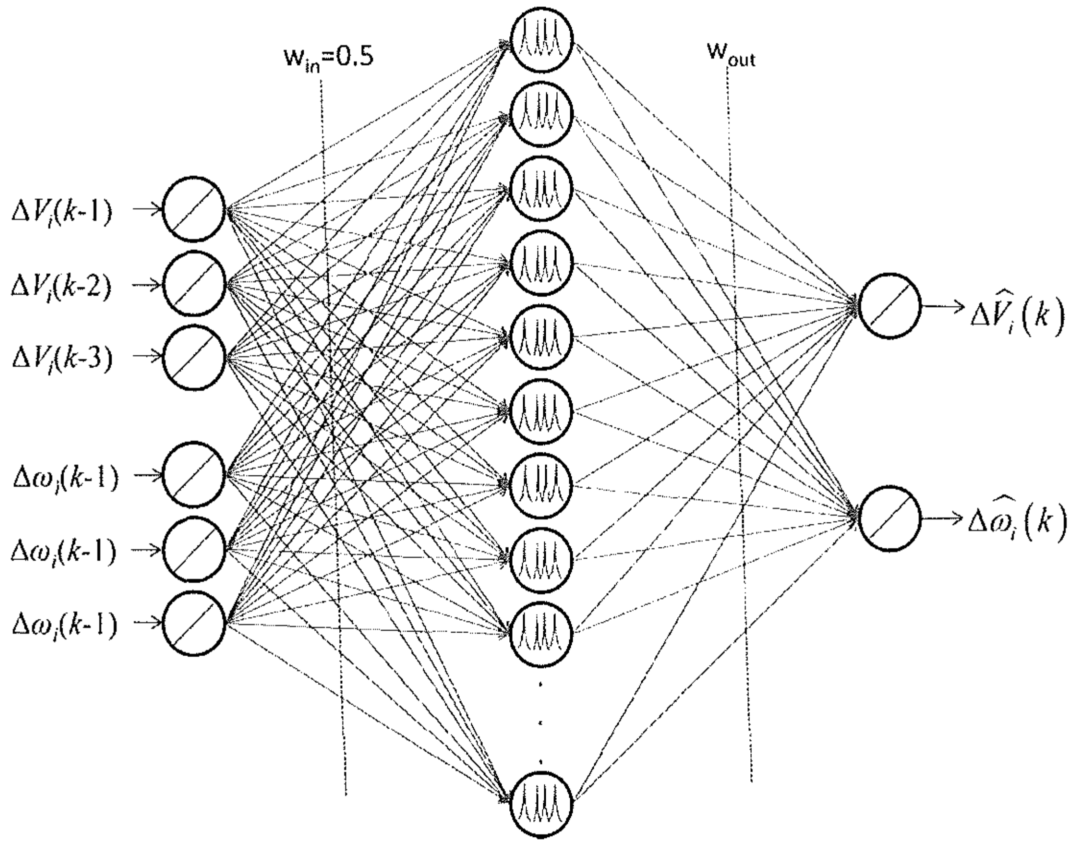


Figure 4.2. Diagram of a feedforward time delay ASNN. All input neurons from input *j* to hidden neuron *i* are set to 0.5, and are not trained.

The ASNN model discussed in this dissertation takes continuous-valued inputs and treats them as a firing rate of spikes received from a presynaptic neuron. The actual spiking of the neuron is never modeled, and is instead abstracted according to (5)-(9). The neuron's final output is given by (9) as the firing rate based on the calculated expected ISI given the input firing rates from all presynaptic neurons. At no point does encoding into spikes nor decoding of spikes occur; the assumed decoding of the ISI is derived directly from the listed equations.

In effect, this is a more complex and potentially more powerful activation function. In its role as a third-generation NN, this ASNN activation function is to sigmoidal activation functions what sigmoidal activation functions were to thresholding functions. Since it takes the same sort of inputs and outputs as second-generation neurons, they can be arranged into the same sorts of networks.

In order to do this, however, the activation function needs to be calculable in reasonable timeframes. The Dawson Integral given in (8) has no analytic solution that does not involve calculating enormous numbers of functional elements in series. However, as shown in Figure 4.3, the Dawson Integral shares its shape and functional properties with the Error Function, or ERF (25).

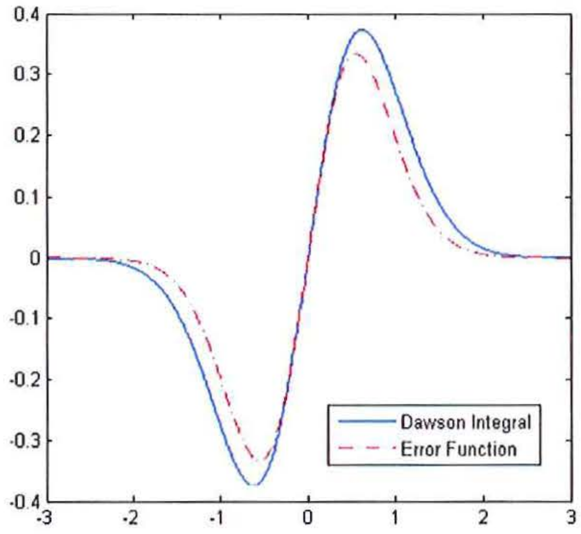


Figure 4.3. Dawson Integral compared to the Error Function. Notice that they differ only by an amplitude; if this amplitude proves a significant difference in functionality, it can be adjusted arithmetically.

$$\operatorname{erf}(x) = \frac{2}{\pi} \int_0^x e^{-t} \mathrm{d}t \tag{25}$$

The ERF does not look any easier to integrate than the Dawson Integral. Unlike the Dawson Integral, however, the ERF has a built-in functional representation in Matlab, and that built-in function operates much faster than does any hand-written function to calculate a Dawson Integral.

With this modification, an ASNN has been implemented in Matlab and can operate in reasonable time compared to an MLP [31]. In order to prepare the ASNN for use in neuroidentification, the centers λ^i around which the hidden neurons were to be clustered must be established as the neurons respond similarly to an RBF. There are a number of possible ways to select these centers. In this experiment, *k*-means clustering is used to find them based on the data sets over which they will be tested. Centers are found for G7's data, G10's data, and for both data sets combined. Figure 4.4 illustrates the differences.

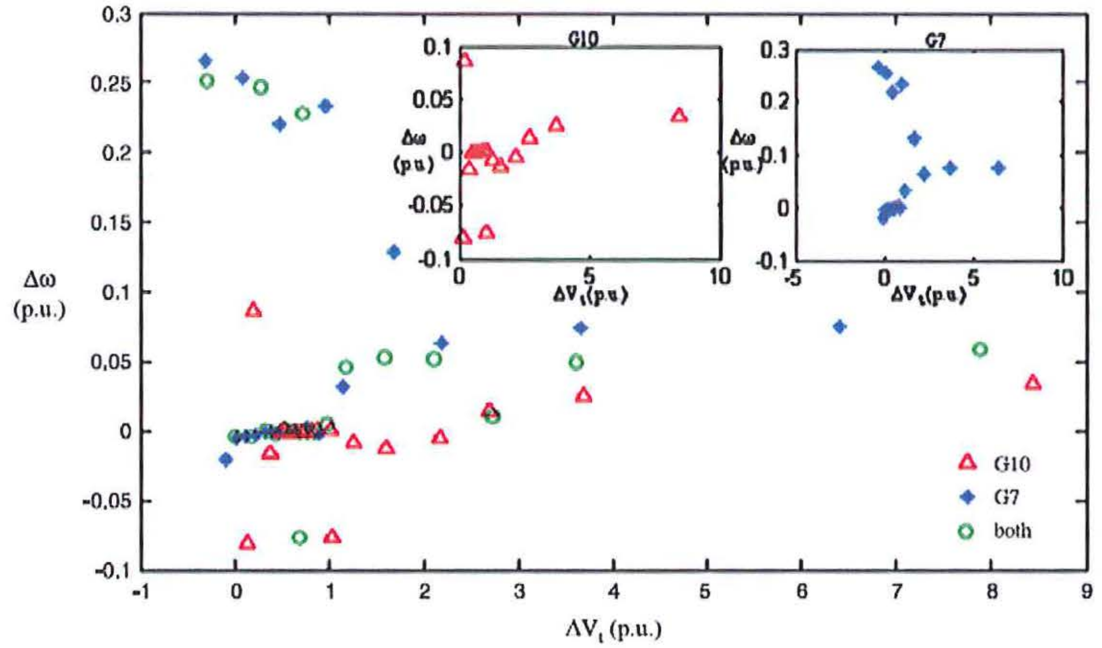


Figure 4.4. Centers for 20 neurons in the hidden layer of an ASNN determined via *k*-means clustering on data samples taken from generator G7 (blue diamonds), generator G10 (red triangles), and combined data samples.

Because the ASNN requires pre-training of λ^i on the dataset to be used, three different sets of centers were trained for comparison. Each generator was tested on centers chosen specifically by its own data set and on centers chosen by the combined data sets of both generators, and their results were compared to each other and to the sigmoidal TDNN's results. As mentioned before, these centers are found via *k*-means clustering in an offline training step on samples of the data.

There are two oscillatory modes into which the generators are driven: "forced" and "natural." Forced perturbations utilize a pseudo-random binary signal (PRBS) to drive the generators through all of their dynamics. This models the changes in load which a real-world power grid would see as customers turn off and turn on devices as they use them. Forced perturbations are constant but predictable based on prior states. Natural faults are the result of temporary disconnects in power lines or even weather-induced literal breaks, and can cause power demands to spike across the surviving routes or shorts across the broken lines. There is no way to predict a natural fault until it happens, as the states of the system pre-fault have very small impact on post-fault states compared to the effect of the fault itself.

The ASNN using centers obtained for Generator G7 is shown on forced perturbation testing in Figure 4.5. Figure 4.6 is the same data set using centers trained on both G7 and G10. Figure 4.7 and Figure 4.8 are G10 with an ASNN whose centers are trained on G10's data and on the unified data, respectively.

Though examination of the terminal voltage prediction error does not seem to indicate any significant drop in fidelity by switching to a more general set of centers based on the combined generator data sets, the speed deviation error climbs to almost 0.2% when the combined data centers are used, versus only a little over half that for the G7 data alone. This is not terribly surprising, as one would expect more specialized centers to be a bit more sensitive, but it is relevant in determining that the choice of centers is significant. Notice, however, the behavior between the two runs with G10 in Figure 4.7 and Figure 4.8.

The %-error for both speed and voltage predictions are notably *higher* for the centers selected based solely on G10's data than for the centers selected based on both generators together, when predicting G10's speed and voltage deviations. This is intriguing, because it implies that the problem is not the difference between the centers and the data set, but rather simply that G10's data set generates less useful centers than G7's. Why this is is unknown, but probably worthy of further examination in works focused more on optimization than testing the ASNN compared to a sigmoidal TDNN.

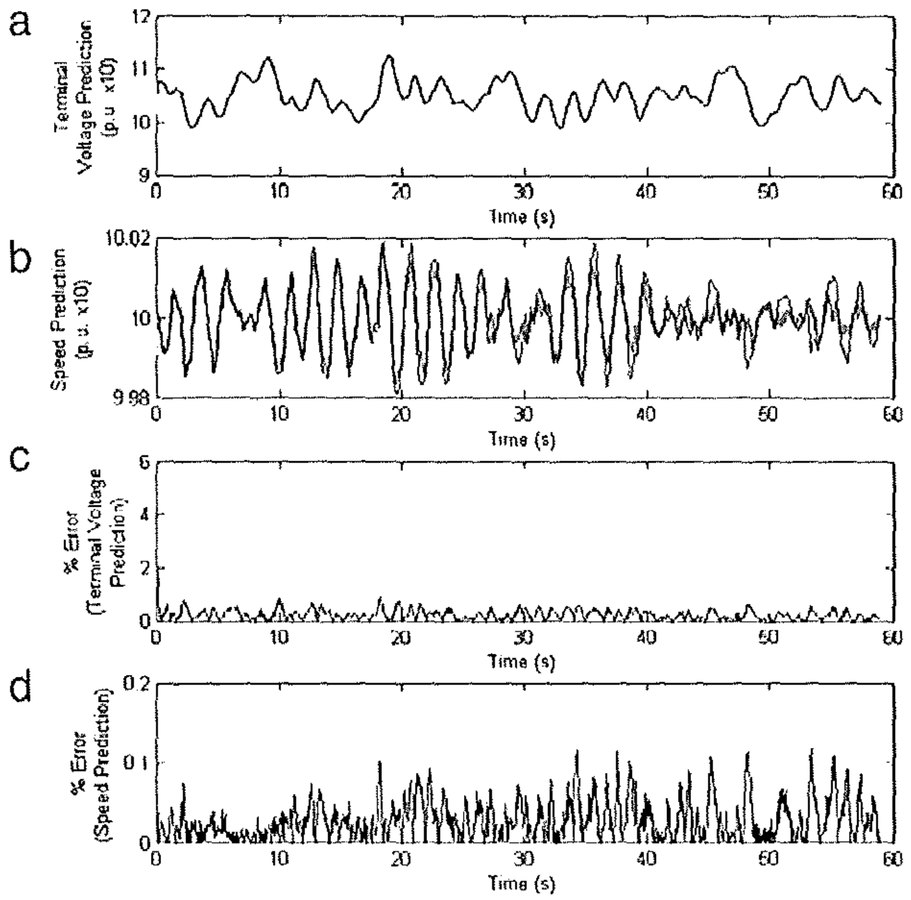


Figure 4.5. Forced perturbation of G7 neuroidentified by an ASNN using centers λ^i trained on G7's data alone. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

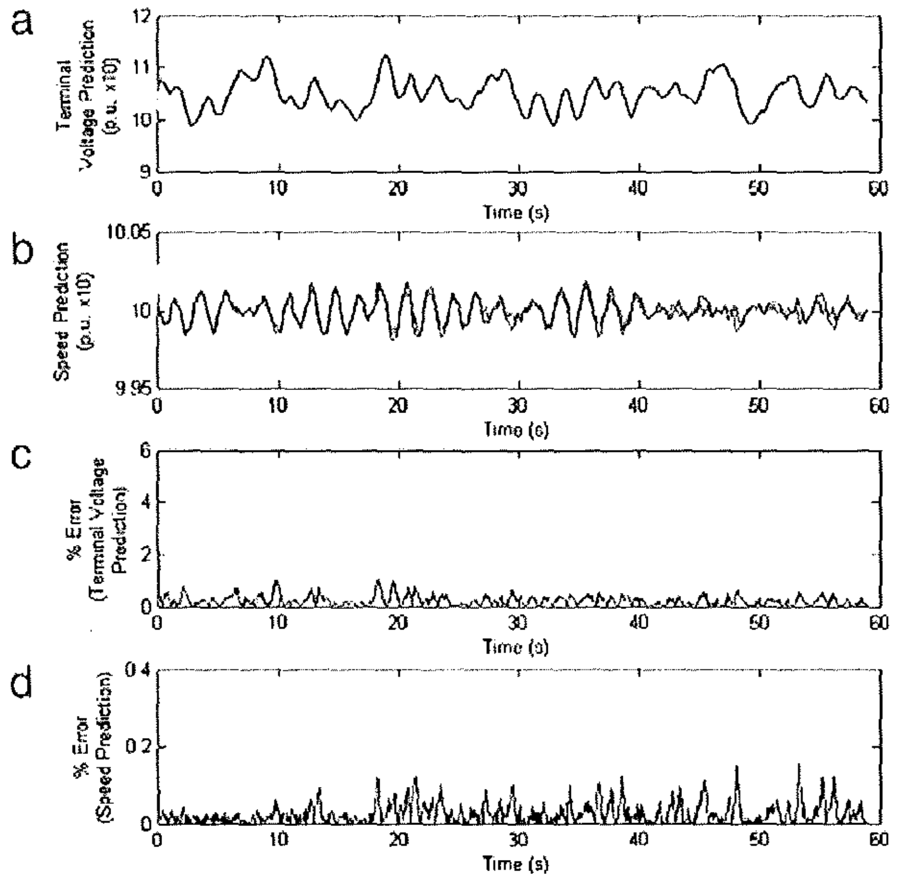


Figure 4.6. Forced perturbation of G7 neuroidentified by an ASNN using centers λ^{i} trained on the combined G7 and G10 data. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

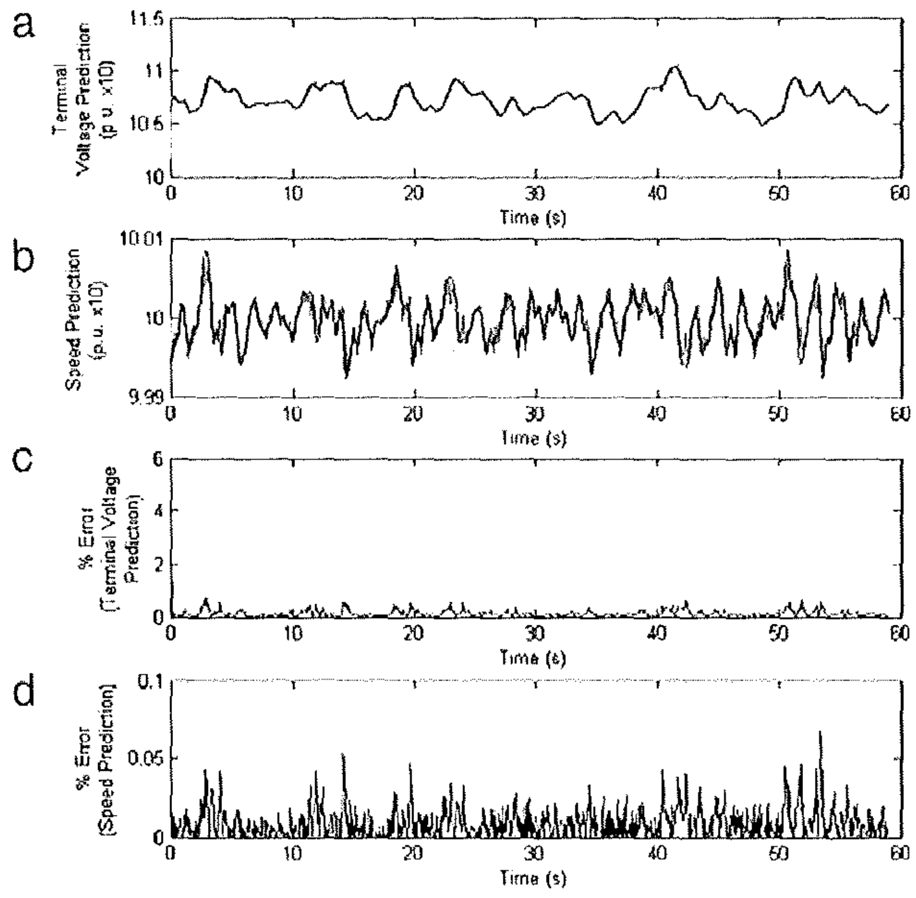


Figure 4.7. Forced perturbation of G10 neuroidentified by an ASNN using centers λ^i trained using G10's data alone. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

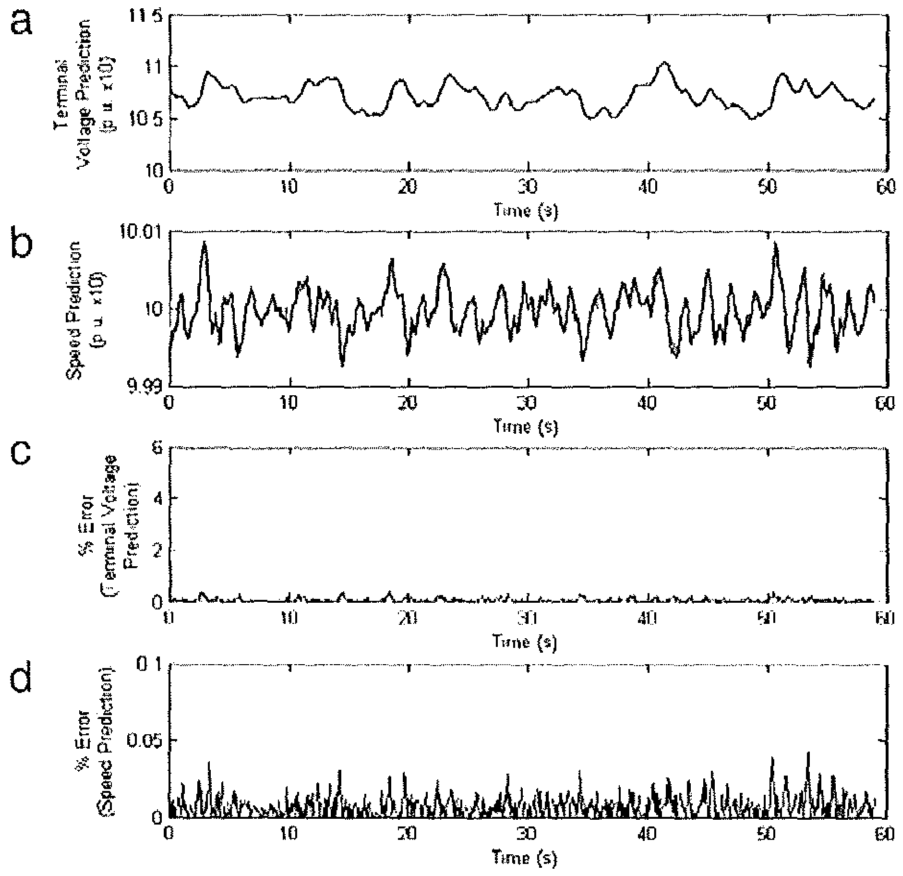


Figure 4.8. Forced perturbation of G10 neuroidentified by an ASNN using centers λ' trained on the combined G7 and G10 data. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

For full comparison, however, it is necessary to also test the ASNN on natural faults. Figure 4.9 tests an ASNN on identifying G7 during a natural fault using its own centers alone; Figure 4.10 tests the same thing with an ASNN whose centers are based on both generators' data. Figure 4.11 and Figure 4.12 repeat these for G10, using its own centers and the combined centers, respectively.

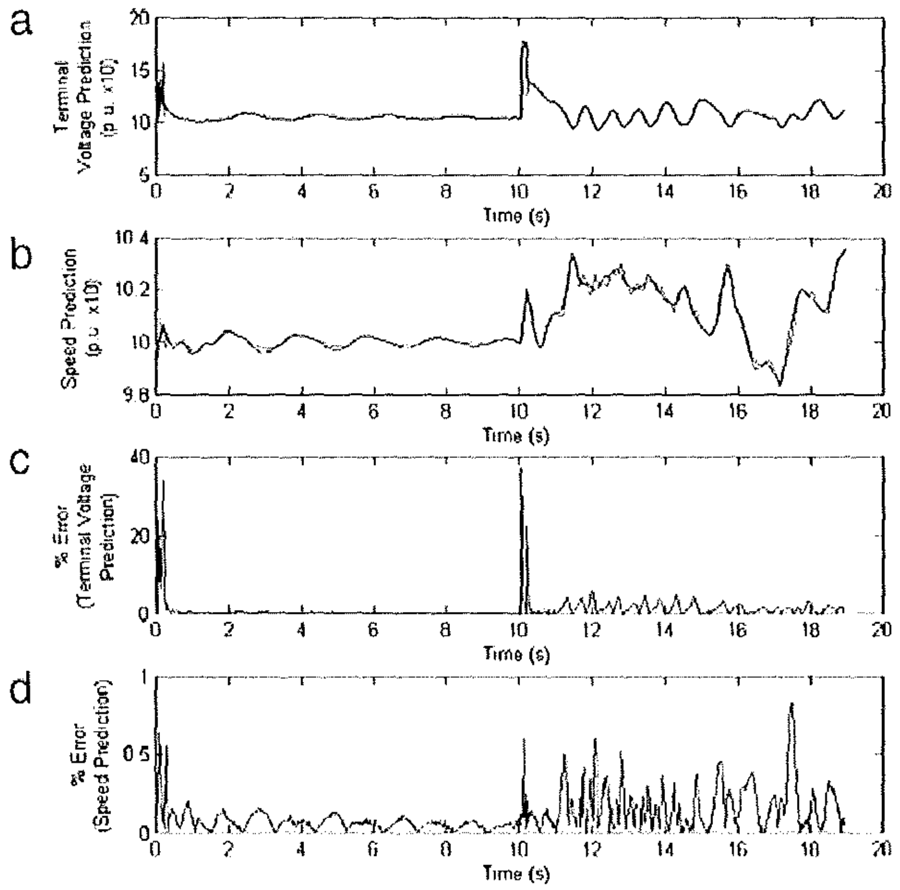


Figure 4.9. Natural fault on G7 neuroidentified by an ASNN using centers λ^i trained on G7's data alone. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

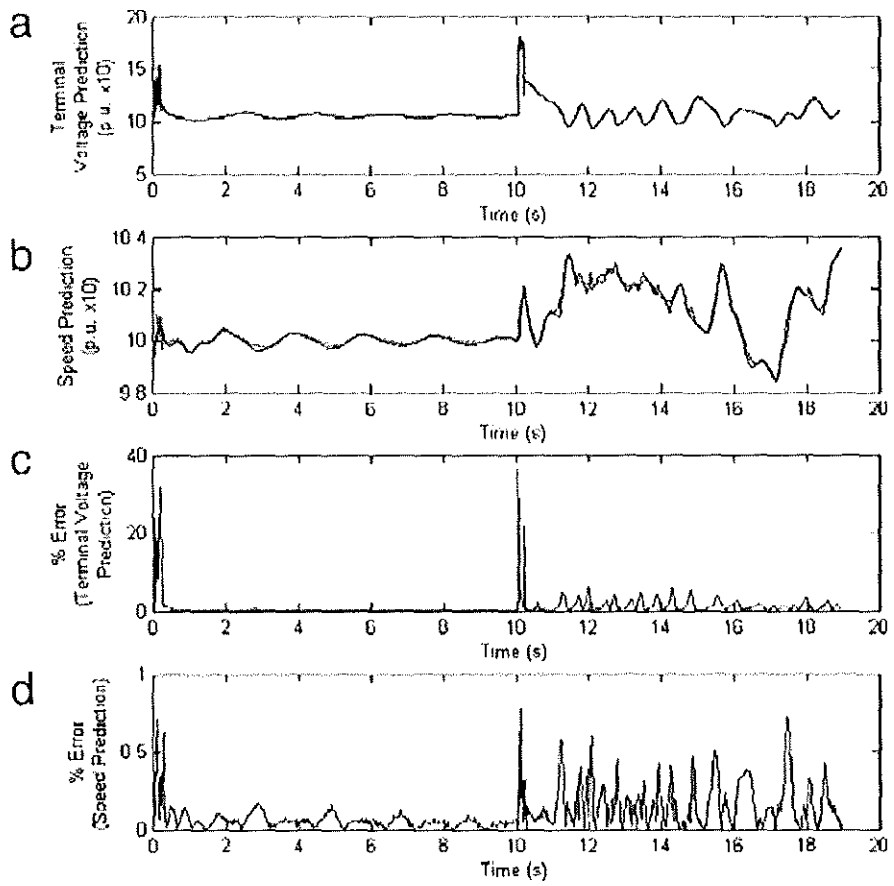


Figure 4.10. Natural fault on G7 neuroidentified by an ASNN using centers λ^i trained on the combined G7 and G10 data. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

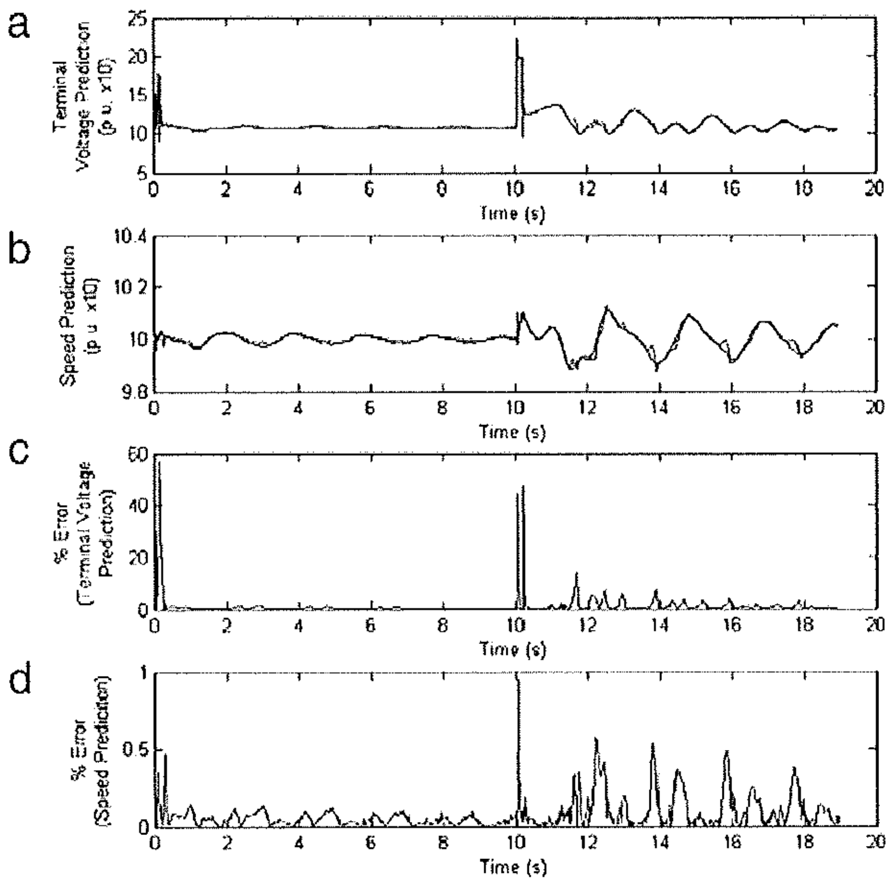


Figure 4.11. Natural fault on G10 neuroidentified by an ASNN using centers λ^i trained on G10's data alone. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

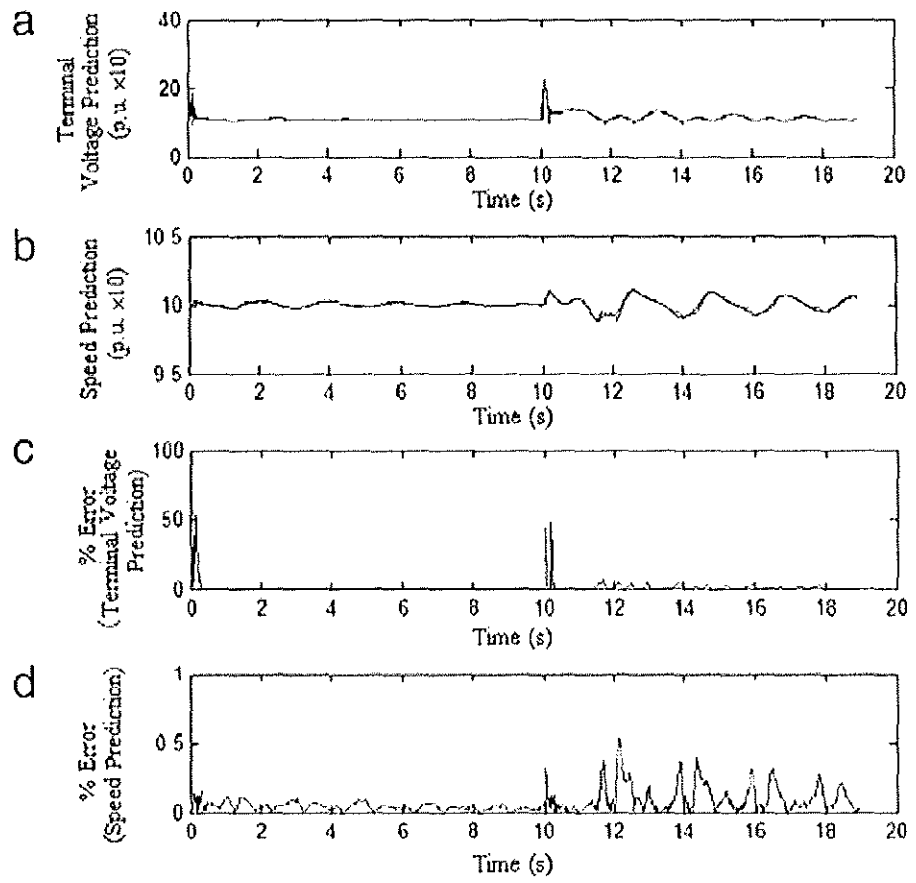


Figure 4.12. Natural fault on G10 neuroidentified by an ASNN using centers λ^i trained on the combined G7 and G10 data. (a) Terminal voltage (thick line) and ASNN estimation (thin line) per unit x10; (b) generator speed (thick line) and ASNN estimation (thin line) per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual generator speeds.

The ASNN does an impressive job of tracking the natural fault. The error is understandably higher than on the forced perturbations, which were generally smaller in magnitude and represented less disturbance to the system. But the complicated activation function used in the ASNN is more computationally demanding than the sigmoidal activation function in a second-generation TDNN MLP. Section 4.4 provides a study of such an MLP on the same data, making it possible to discern whether the extra computational power per neuron is warranted.

4.4. MLP NEUROIDENTIFICATION

To establish a baseline for comparison, the IEEE 10 machine 39 bus power system shown in Figure 4.1 was also tested on a feedforward TDNN using sigmoidal neurons in the hidden layer. Figure 4.13 illustrates the TDNN architecture with the sigmoidal neurons. The same number of neurons are used in the hidden layer of the MLP as in the ASNN, and both are exposed to the same data sets of forced perturbations and natural faults.

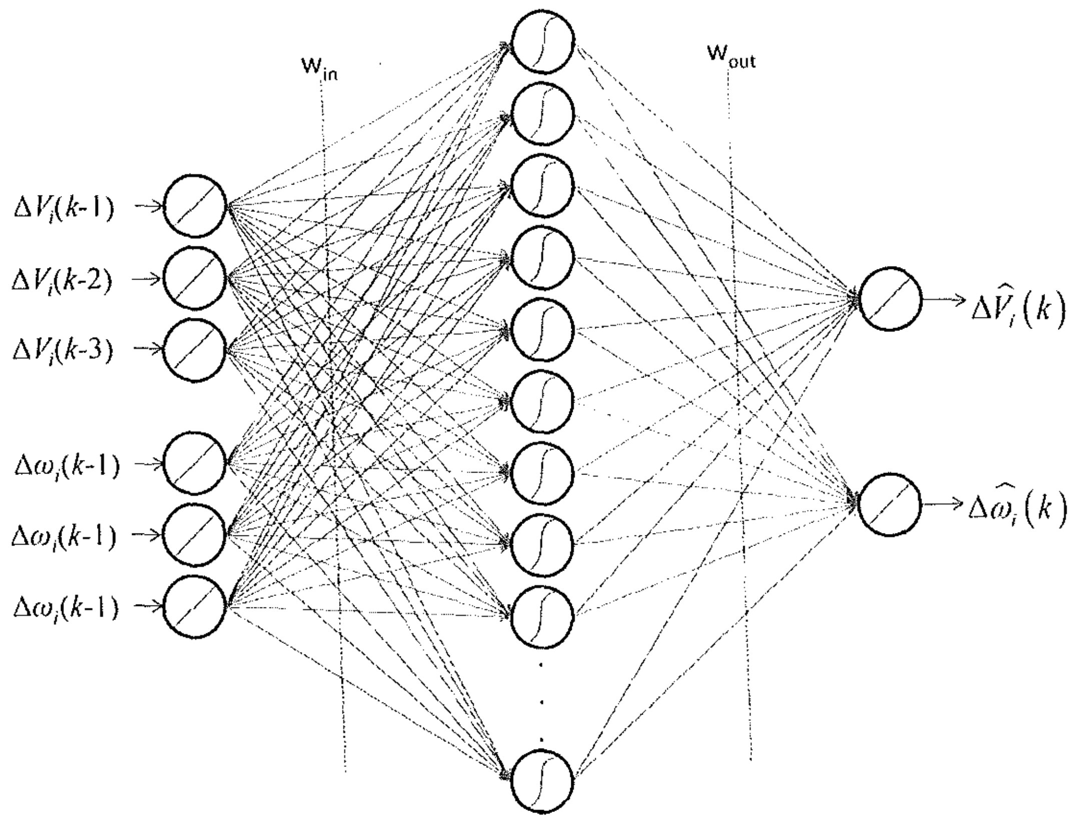

W_{in}
W_{out}
$\Delta V_i(k\text{-}1)$
$\Delta V_i(k\text{-}2)$
$\Delta V_i(k\text{-}3)$
$\Delta\omega_i(k\text{-}1)$
$\Delta\omega_i(k\text{-}1)$
$\Delta\omega_i(k\text{-}1)$
$\Delta\hat{V}_i(k)$
$\Delta\hat{\omega}_i(k)$

Figure 4.13. Diagram of a feedforward time delay ASNN. All synaptic weights are trained via gradient descent based backpropagation.

The input and output weights are trained via gradient descent based backpropagation. Other than this, where appropriate, the parameters are set the same as in the ASNN as shown in Figure 4.1. Figure 4.14 plots the terminal voltage and speed deviation of G7 under forced perturbation. The thick line is the actual deviations recorded after the fact. The thin line is the predicted values generated before the actual values were received. They are overlaid to show how close the MLP came in identifying them. The %-error for each is calculated according to (26). Figure 4.15 plots the same data for G10.

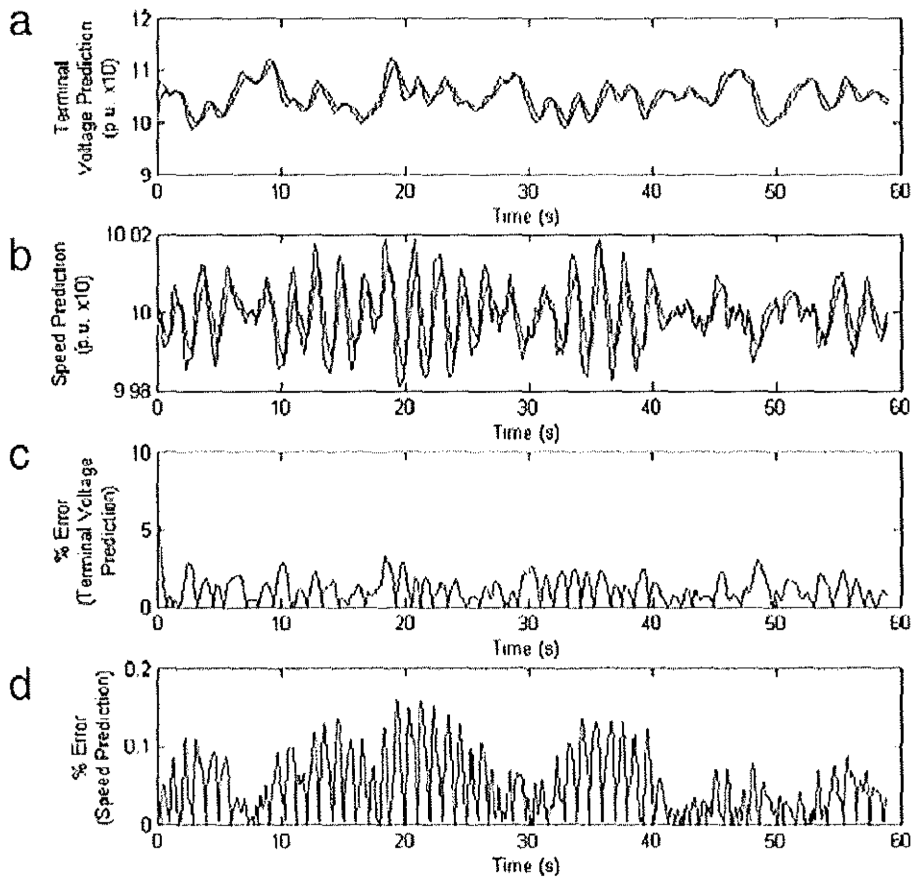


Figure 4.14. G7 forced perturbation of a sigmoidal TDNN. The signals and their estimations are scaled up by a factor of 10 to allow the neuroidentifier a palatable scale. (a) Terminal voltage estimated (light line) and actual (heavy line) per unit x10, (b) generator speed estimated (light line) and actual (light line) per unit x10, (c) % error between estimated and actual terminal voltage, (d) %-error between estimated and actual generator speed.

$$\%error = \left|\frac{actual - estimated}{actual}\right| \times 100\% \qquad (26)$$

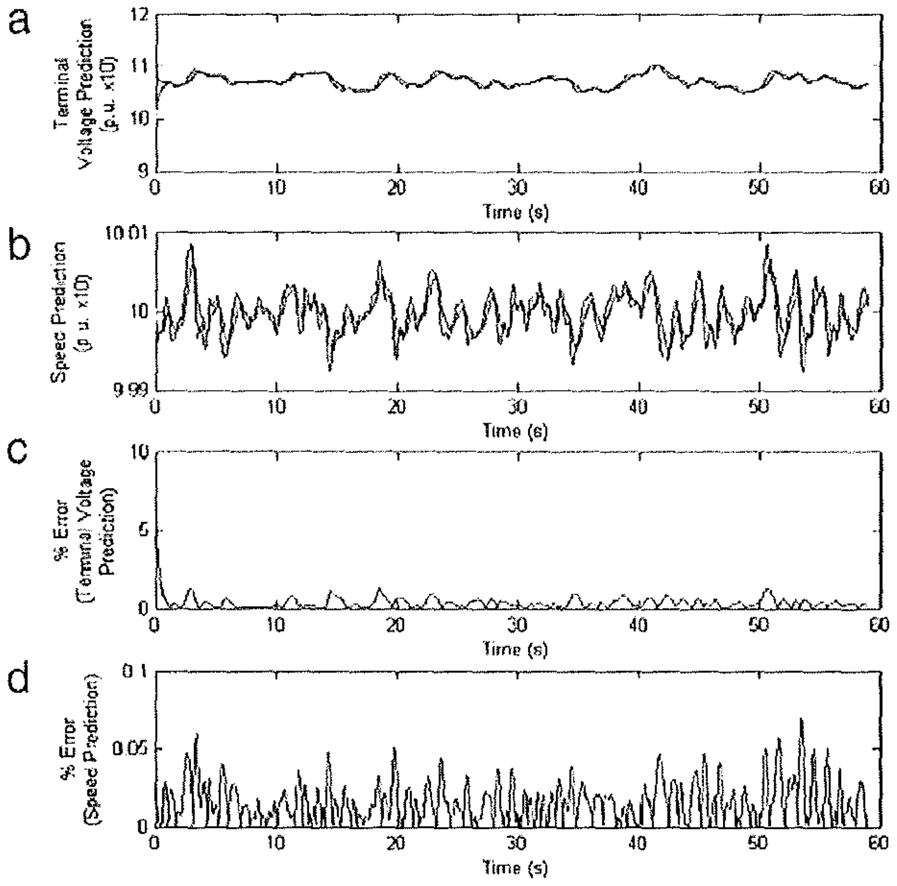


Figure 4.15. G10 forced perturbation of a sigmoidal TDNN. The signals and their estimations are scaled up by a factor of 10 to allow the neuroidentifier a palatable scale. (a) Terminal voltage estimated (light line) and actual (heavy line) per unit x10, (b) generator speed estimated (light line) and actual (light line) per unit x10, (c) % error between estimated and actual terminal voltage, (d) %-error between estimated and actual generator speed.

The natural faults are applied to G7 and G10 in two steps. At 0s on the plots in Figure 4.16 and Figure 4.17, a single-line phase-to-ground fault is applied, after which the system is allowed to recover for 10s. Thereafter, a three-line phase-to-ground fault is applied, and the effects observed for another 10s. Notice that the sigmoidal TDNN has difficulty immediately following a fault, as there was no way it could predict it. For the single-phase fault, the sigmoidal TDNN is relatively able to continue to track and predict the oscillations, but once the three-phase fault is applied, the sigmoidal TDNN struggles to remain within the same ballpark, with errors approaching 20% in its predictions.

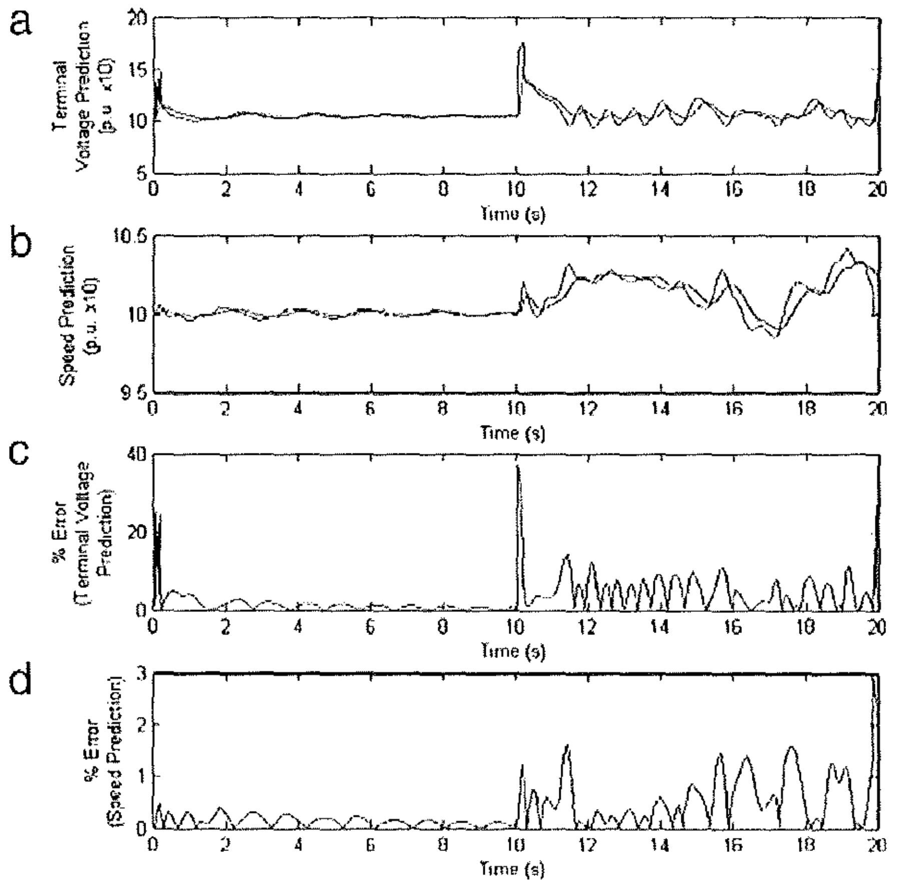


Figure 4.16. Natural fault applied to G7, with a single phase fault at 0s and a three phase fault at 10s. The signals (thick line) and their estimations (thin line) are scaled by a factor of 10 to allow the neuroidentifier an palatable scale. (a) Estimation of terminal voltage per unit x10; (b) estimation of generator speed per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual speeds.

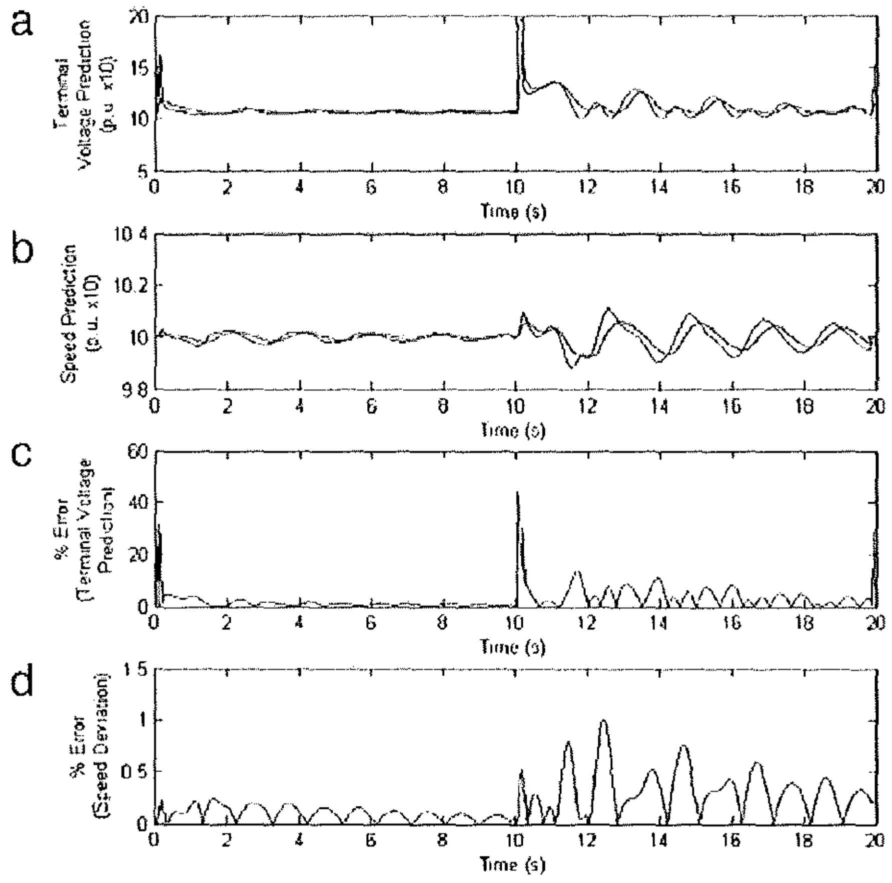


Figure 4.17. Natural fault applied to G10, with a single phase fault at 0s and a three phase fault at 10s. The signals (thick line) and their estimations (thin line) are scaled by a factor of 10 to allow the neuroidentifier an palatable scale. (a) Estimation of terminal voltage per unit x10; (b) estimation of generator speed per unit x10; (c) %-error between estimated and actual terminal voltages; (d) %-error between estimated and actual speeds.

The sigmoidal TDNN is capable of tracking and predicting generator dynamics under forced conditions and even under limited fault conditions, and is visibly closer than simply assuming everything remains nominal. However, there is definitely room for improvement. The natural fault neuroidentification is where the power of the ASNN compared to the sigmoidal TDNN. Not only is the error less than half that of the sigmoidal TDNN in the ASNN for speed prediction, but just visually, it's clear that the ASNN does a significantly better job of recovering and tracking the speed and voltage data after a massive fault.

Initial examination of the sigmoidal TDNN's performance on forced perturbations compared to even the ASNN which has the worst performance shows marked improvement. It is all but impossible to see the difference between the predicted and actual voltages and speeds, and the %-error is consistently lower on the ASNN runs than on the MLP. The ASNN, simply by using an activation function that abstracts spiking behavior, is already a major improvement over the second-generation sigmoidal TDNN. What, then, might be possible with a BSNN?

4.5. BSNN: POLYCHRONOUS SPIKING NETWORK

The PSN described in Section 3 has already been demonstrated to be able to reproduce a relatively simple variable-frequency sine wave. Now, it is to be tested on a power system problem like that in the previous section. Because the PSN requires temporal dependence in its data input stream to operate successfully, a longer stream of power system data is generated. A slightly simpler two-area, four machine power system shown in Figure 4.18 is used for this experiment. All four generators' speed deviation $d\omega$ and voltage deviation dV were generated by simulation for 255 seconds on the RTDS at the RTPIS Laboratory. Figure 4.19 shows the two inputs to each of the four generators. However, the scale of those inputs is so varied that it would be difficult for any ANN to discern all of their dynamics, so they need to be normalized as shown in Figure 4.20.

The PSN framework used is the same as in Section 3. Figure 4.21 illustrates the data flow from the original generator one signal (taken from 10402 centiseconds to 11601 centiseconds) which gets normalized and passed through the GRF encoder to form input spikes on 24 sensory neurons (twelve sensory neurons per input), into the PSN, through the decoder, and out to produce the final output. The PSN uses 1000 Izhikevich neurons, and the decoder uses 20 hidden neurons. Normalization of the range of input ensures that the full dynamical range of the PSN is stimulated by the data presented. The task given to the PSN here is to learn the speed deviation dynamics of the time frame stated above (roughly 10.5 to 11.6 s) in generator 1, using that generator's speed and voltage dynamics as its inputs.

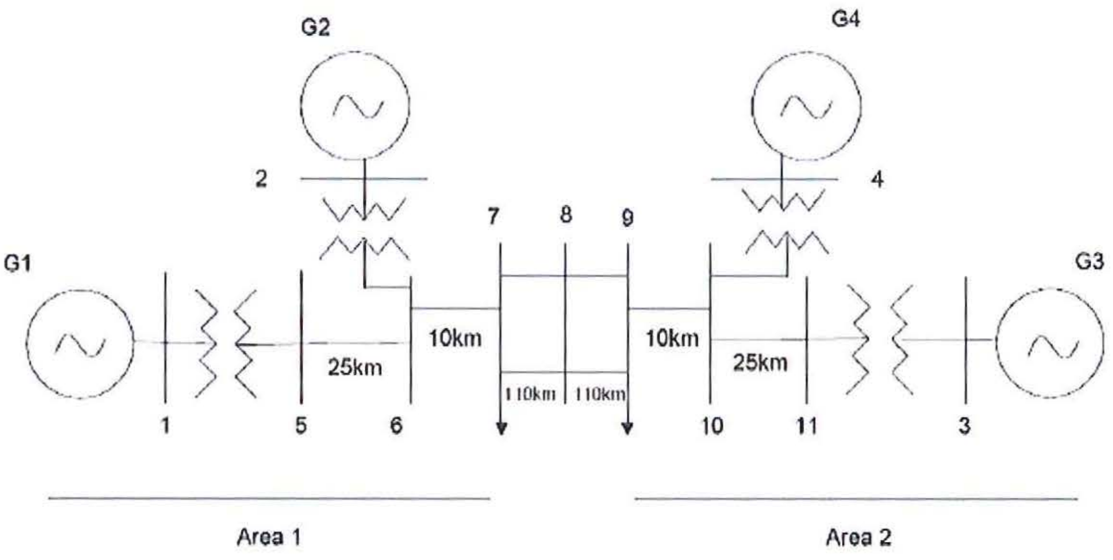


Figure 4.18. Two-area four-machine power system simulated on the RTDS [21].

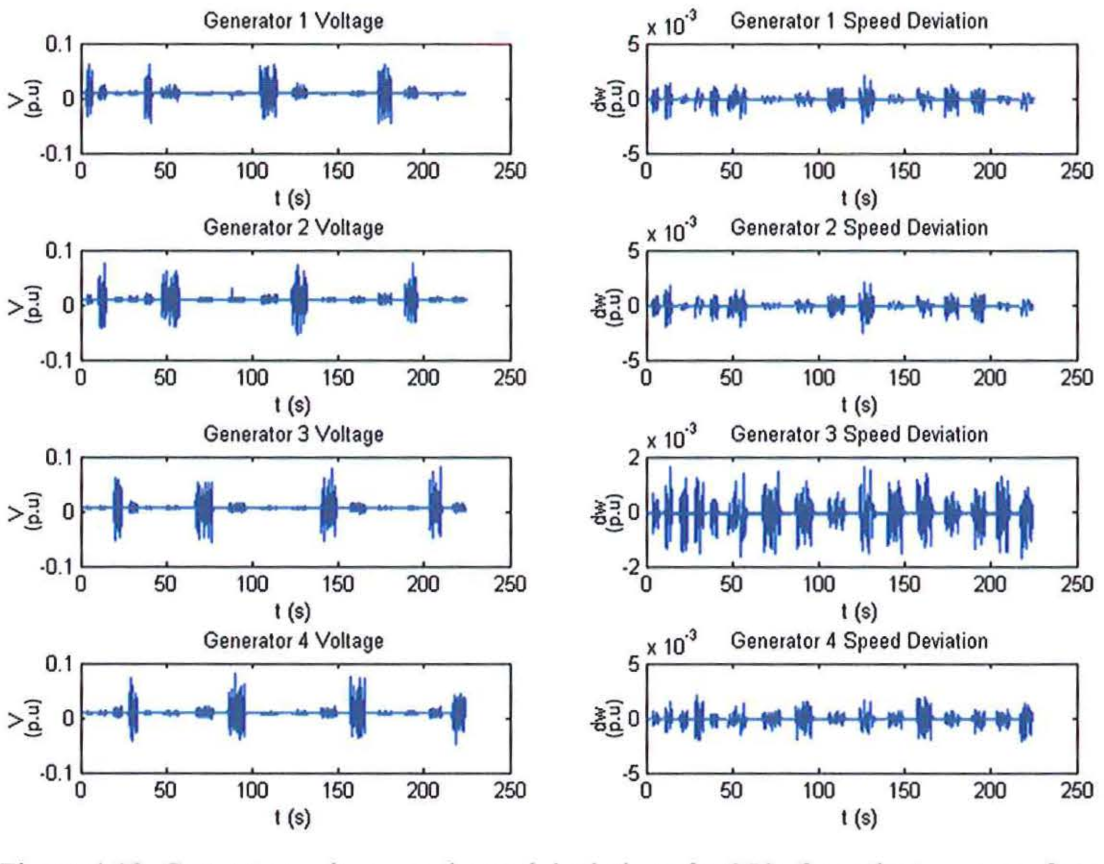


Figure 4.19. Generator voltages and speed deviations for 250s from the two-area, four-machine power system shown in Figure 4.18.

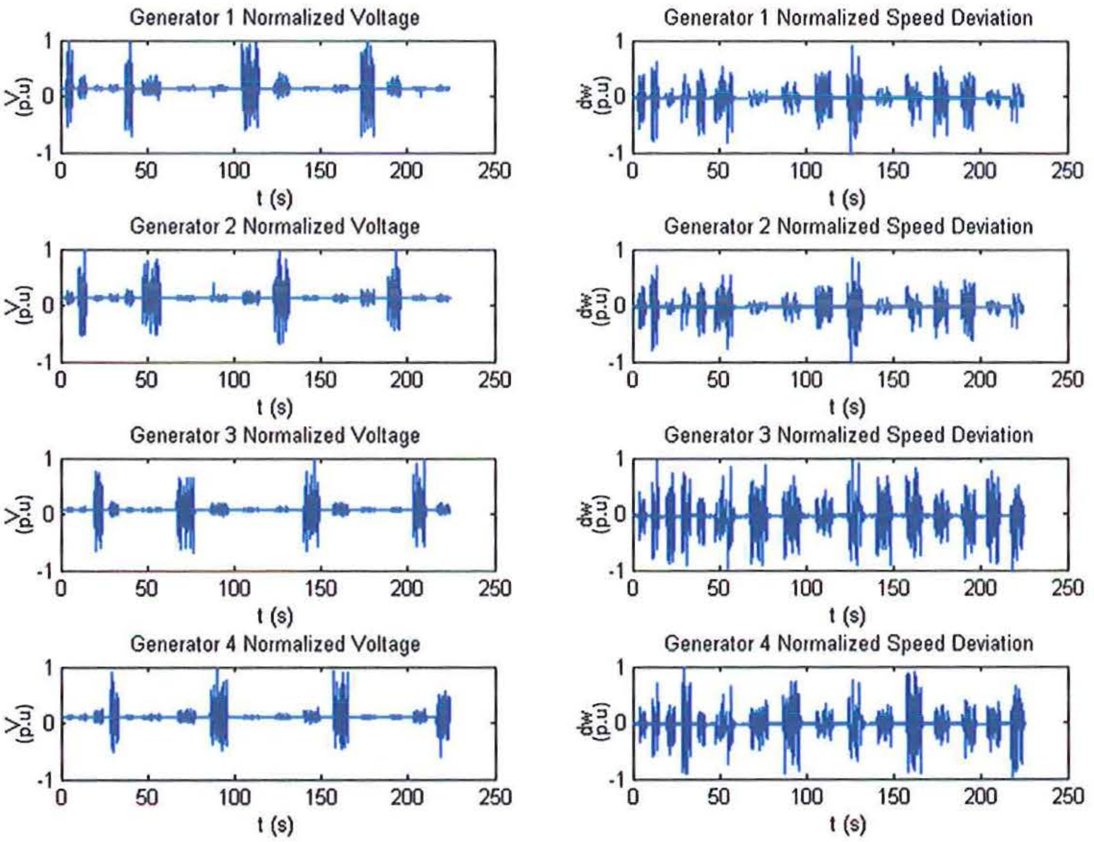


Figure 4.20. Normalized values from the four generators, used as inputs to the PSN.

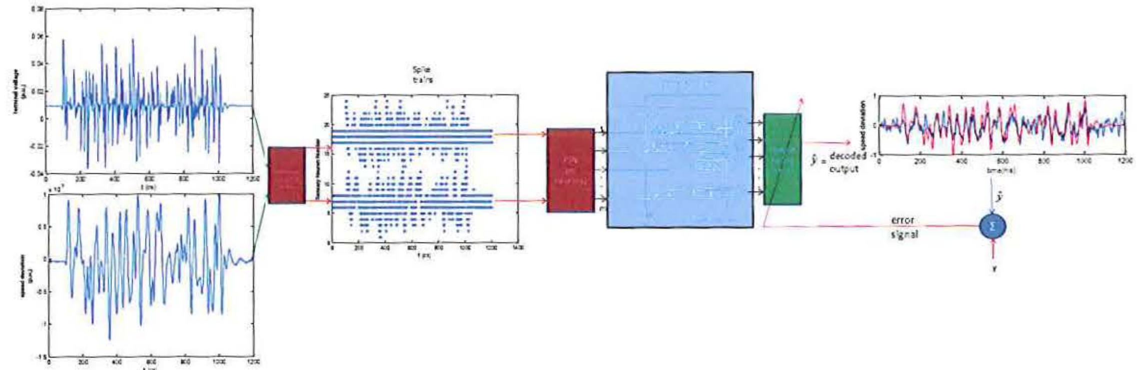


Figure 4.21. Illustration of data flow through the PSN framework. Normalization of the inputs is left out of the diagram for space reasons, and happens before they are entered into the GRF encoder.

For training purposes, the selected time frame of generator 1's activity is presented repeatedly to the PSN. Several training methods have been attempted to tune the decoder, including online backpropagation training, offline batch backpropagation training, and PSO-based training. The PSO-based training provided the best results, shown in Figure 4.22.

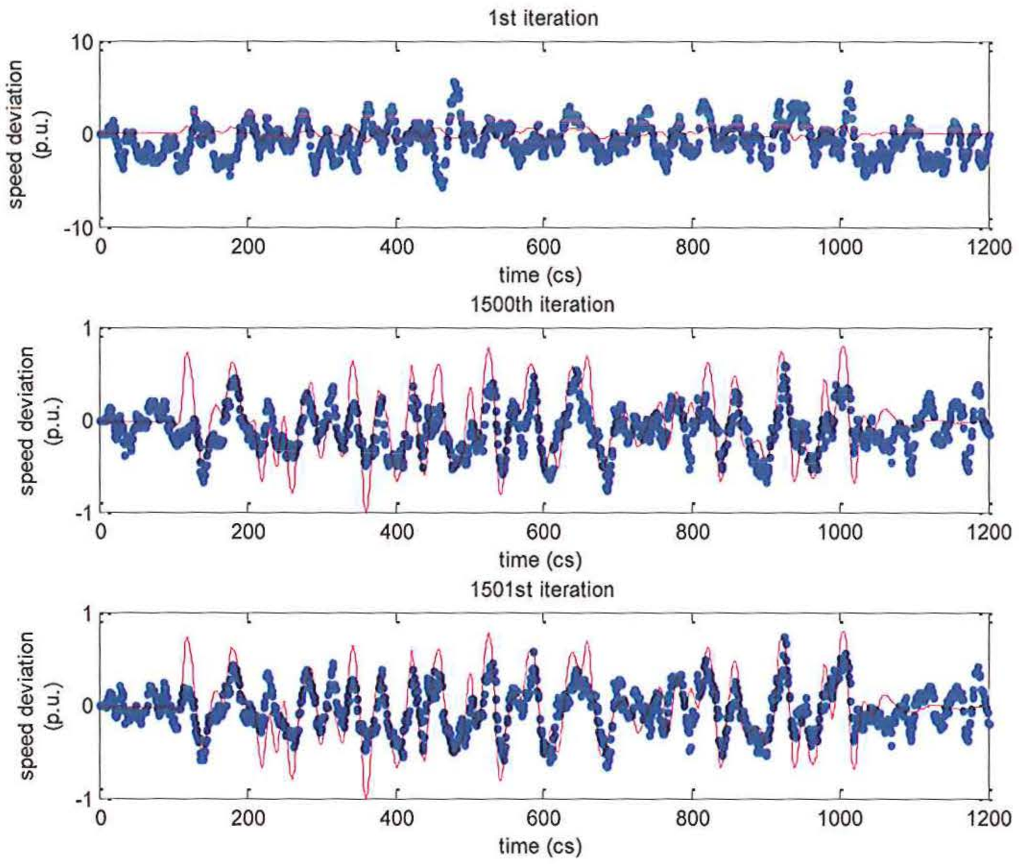


Figure 4.22. PSO-based training results of a 1000-neuron PSN on generator 1. Inputs were generator 1's current voltage and speed deviations (normalized), and the target (shown in red in the figure) is the current-state speed deviation. This has to overcome a 10 cs computation time to perform current-state neuroidentification. The blue dots are the output signal. The 1501st iteration of the PSO-based decoder tuning, shown above, had a mean squared error of only 0.055.

The output follows the target less well than the ASNN outputs in the prior subsection, and is not even as good as the BSNN results on (16) in Section 3. However, it is clearly following the presented pattern, which is not possible without the PSN possessing relevant dynamics. The PSO training is batch by nature, so it is impossible that the decoder weights could be slaved on a moment-to-moment basis to track the outputs; they must have isolated weights which can at least extract the dynamics across the full input stream. Figure 4.23 shows mean squared error of the best performing weight set for all 1501 iterations of the one-second loop.

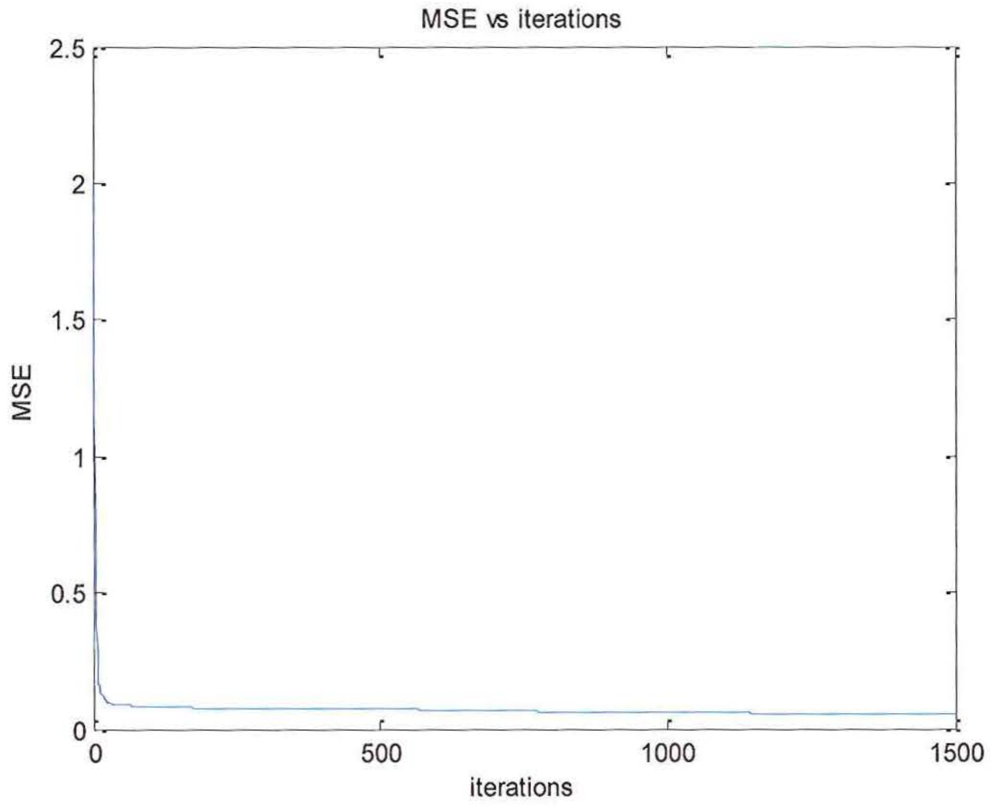


Figure 4.23. MSE of the best performing particle in the PSO training the decoder on the data used to generate Figure 4.22.

4.6. SUMMARY

Applying ANNs to identification and prediction of power system dynamics brings, a powerful computational tool to bear on a highly nonlinear and important control problem. Compared to classical linearized controllers, the adaptability of ANNs offers the ability to run reliably even as a system moves away from the nominal operating point. To adapt to changing environments, neuroidentification of generator dynamics (voltage and speed deviations in particular) is necessary. This Section outlines two experiments in SNN-based neuroidentification: one using an ASNN on two generators in the IEEE 39-bus system and comparing the results to a traditional sigmoidal MLP on the same; and another attempt with a two-area multimachine power system to implement a BSNN neuroidentifier on one of the four generators' speed deviations using both voltage and speed deviation as inputs from all four generators.

The input and target data for all experiments in this section were generated using the RTDS at the RTPIS Laboratory. The results for the ASNN compared to the MLP demonstrate that the ASNN is not only able to operate on the problem, but to do so with more fidelity than the MLP. The results for the BSNN are not as impressive as even the sigmoidal MLP, let alone the ASNN; however, the BSNN demonstrably possesses and has a decoder which can extract the dynamics of the generator signals on which it is trained. Additionally, the advance of the activation function as inspired by spiking neurons and their demonstrated superiority to sigmoidal neurons of the previous generation. These are two solid steps forward for the third generation of neural networks as a serious tool for real-world use.

5. SUMMARY

5.1. INTRODUCTION

The advancement of ANN research has reached its third generation. The first was ADALINE, the McCulloch-Pitts neuron [3], and the Hodgkins-Huxley neural model [2], with their binary threshold activation functions and their integrate-and-fire models that were of use primarily to biological neurological study [1]. The second generation took off with the introduction of backpropagation through time in Werbos's work [4], [5] and the development of more continuous activation functions (such as the popular and by-now traditional sigmoidal function) [6]. As the second generation progressed, new architectures developed with increasingly complex recurrence.

Now, as the third generation utilizing spiking neurons as viable ANN components is developing, it is important to understand its strengths and weaknesses compared to prior generation models. This dissertation has discussed areas in which SNNs are already competent – pattern recognition, logic gate functionality, image and sound processing, etc. – and has detailed several needed advances towards making them stand along side second-generation ANNs as universal continuous function approximators and time-series identifiers.

5.2. CONTRIBUTIONS

To use a BSNN on continuous data, it is essential that the data be encoded in a manner that ensures the resulting spike stream contains the original information. It also must be capable of outputting spike streams which can be meaningfully decoded into continuous values. The spiking neural model and the neural architecture are also important. Spiking neurons work as well as they do in nature because of how they interconnect; without that, much of their computational power and reason for being used is lost.

It is with this eye towards functionality that two kinds of SNN are studied: one which uses traditional feedforward architecture and an abstracted model of spiking neurons as a new sort of activation function in place of the sigmoid used in many second-generation models; and one which uses biologically-inspired Izhikevich neurons in a highly recurrent dynamic reservoir.

The specific contributions and accomplishments made in this dissertation are:

- development of a novel encoding method (DNNR) for converting arbitrary continuous-valued data into spike streams for input into BSNNs
- detailed algorithms for that method and one other (GRF)
- development of reversing algorithms to recover data from spike streams created by those two methods without needing prior knowledge of what data was encoded
- analysis of these encoding methods and a third (PREM) to determine that the encoded data is present in the spike stream by recovering it from the spikes
- demonstrated how polychrony can enable a PSN to have more data patterns than there are synaptic connections even in a fully connected network
- development of a decoder which translates PSN output spikes into meaningful continuously-valued functional outputs
- demonstration of the PSN's capability to learn a time-series sine wave using this decoder
- use of the ASNN that treats the abstracted spiking neuron as an activation function in a traditional TDNN on a power system identification problem and compare it to an MLP
- demonstrated a BSNN framework capable of neuroidentification of generator dynamics to a degree of fidelity sufficient to prove that a BSNN contains the dynamics and that they can be extracted.

5.3. SECTION SUMMARIES

5.3.1. Spiking Neural Networks. The SNN is the third generation of ANNs. This dissertation started by reviewing the state of the art in SNN modeling along with its predecessors. A primary focus was given to the Izhikevich model of spiking neurons for biologically faithful modeling of neuron behavior. An abstracted model of spiking neurons as an activation function in a more traditional feedforward style network is also examined for its ability to handle inputs and outputs as easily as second-generation sigmoidal activation function neurons.

The hurdles facing SNNs as a useful ANN are outlined, primarily the need for biological neurons to have an encoder and a decoder which can handle continuous numbers and to have a highly recurrent architecture. The ASNN is proposed as a first effort to overcome the encoding issue, as it is capable of taking numeric input and provide numeric output without any encoding required.

5.3.2. BSNNs and Applications Thereof. The primary contribution of this work is the framework for a BSNN which can handle continuous-valued inputs and provide continuous-valued outputs. Several encoding methods have been examined and algorithms for two – DNNR and GRF – have been provided [30]. The Appendix contains a flow chart explaining the third method [27].

One problem the author found with most encoding methods was the simple assumption that the spike trains generated thereby actually contained the original data. This dissertation also provides algorithms for reversing the encoding of DNNR and GRF, and thus proves that the spikes generated by these methods contain the data desired [30].

For the main "body" of the BSNN framework, the PSN architecture backed by the Izhikevich neuron was chosen. The property of polychrony has been demonstrated to be able to generate more data patterns than there are synapses in the system, enabling a PSN to have ever-increasing granularity in its pattern identifications as more neurons and thus more synapses are added. The explosion of possible unique outputs which respond to unique inputs results in a continuum of possible outputs resembling that of modern floating point numbers. Add in the novel Gaussian function-based decoder which converts spike patterns to continuous-valued numeric outputs and an MLP-based tuner to isolate desired dynamics, and the framework presents a complete capacity to handle arbitrary numeric function approximation, just like prior generation ANNs.

The framework's triumph is the demonstration of its ability to reproduce a variable-frequency sine wave as given by (16), both with online training and with the decoder's weights frozen. It truly learned a continuous-valued function. It needs time-series information, however, as the PSN is a temporal system and does not learn individual inputs as easily as it does those whose sequences have meaning.

5.3.3. Neuroidentification of Power Systems with SNNs. Dynamic power system identification is a real-world problem to which ANNs have been successfully applied in the past [52]. These are, however, systems which run up against the limits of second-generation ANN capabilities. It is hoped that SNNs may be able to overcome some of these limitations, and the polychrony property of ever-increasing dynamic patterns is a promising step in the right direction. However, before SNNs can be compared to their predecessors to see if they can overcome their limitations, they must first be shown to be able to handle the problem at all.

The ASNN is tested on a multimachine power system first, and its results compared to a second-generation time-delay MLP. The results are actually very favorable. It requires a bit more set-up, as it resembles an RBF network in its need for "centers" for the neurons' activities, but it ultimately out-performs an MLP with the same number of hidden neurons.

The BSNN test on a similar multimachine power system was less obviously impressive, learning to track the signal with less fidelity than the ASNN or even the sigmoidal MLP. However, that it was capable of demonstrating any tracking of the generator dynamics indicates that the BSNN does have the ability to operate on continuous, arbitrary-valued functions! This great hurdle of capability demonstration having been conquered, it now remains to develop better ways of developing the framework which makes translation from continuous-valued regimes into the world of spiking information and back smoother and more powerful. It is no longer a question of *if* a BSNN is capable of modeling continuous-valued functions. Having established that it *can* be used on similarly broad problems as its second-generation predecessors, it is now possible to begin studying whether its computational capabilities are greater.

5.4. SUMMARY

The development of a BSNN that is capable of continuous-valued function approximation is very exciting, as it represents the first time (to the author's knowledge) that this has been done with faithfully-modeled spiking neurons in a highly-recurrent architecture. While this technology is more fully developed, the ASNN has proven to be a viable contender with its second-generation cousins on real-world neuroidentification problems. The third generation of ANNs has been under investigation for more than a decade, and this dissertation presents work that enables SNNs to operate on theoretically any problem the second-generation ANNs could handle.

5.5. SUGGESTED FUTURE DIRECTIONS

The current limitations of the BSNN framework presented here are an inability to handle functions which do not have a time-series component, and difficulties training them on highly complex functions such as the power system identification problem presented in Section 4. More immediately for the ASNN, the chief limitation of second-generation ANNs which inspires the need to look to third-generation ANNs to tackle problems such as those presented in Section 4 is one of scalability. Though polychrony has been demonstrated for the BSNN, it is not a property of the ASNN. Further, it has not been truly proven to solve this scaling problem even in a BSNN.

It is therefore suggested that future work pursue:

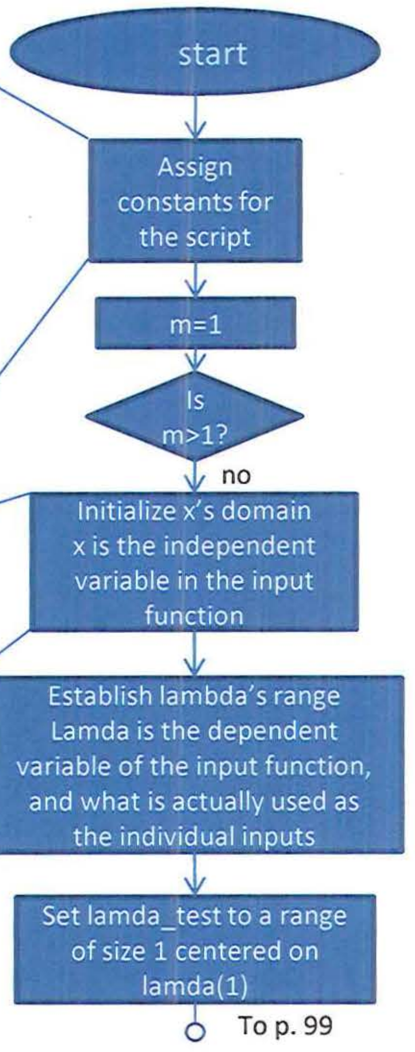
- incorporate a time-series element to static, randomly-ordered inputs, perhaps presenting them repeatedly or with sufficient time between stimuli to allow the response from the previous stimulus to die out in an effort to achieve time-independent function approximation. This would actually be similar to the behavior abstracted in the ASNN.
- test the ASNN on problems where the scaling issue is known to be a problem for second-generation ANNs to see if it can handle these complex problems with fewer neurons and less computational effort than its second-generation counterparts.

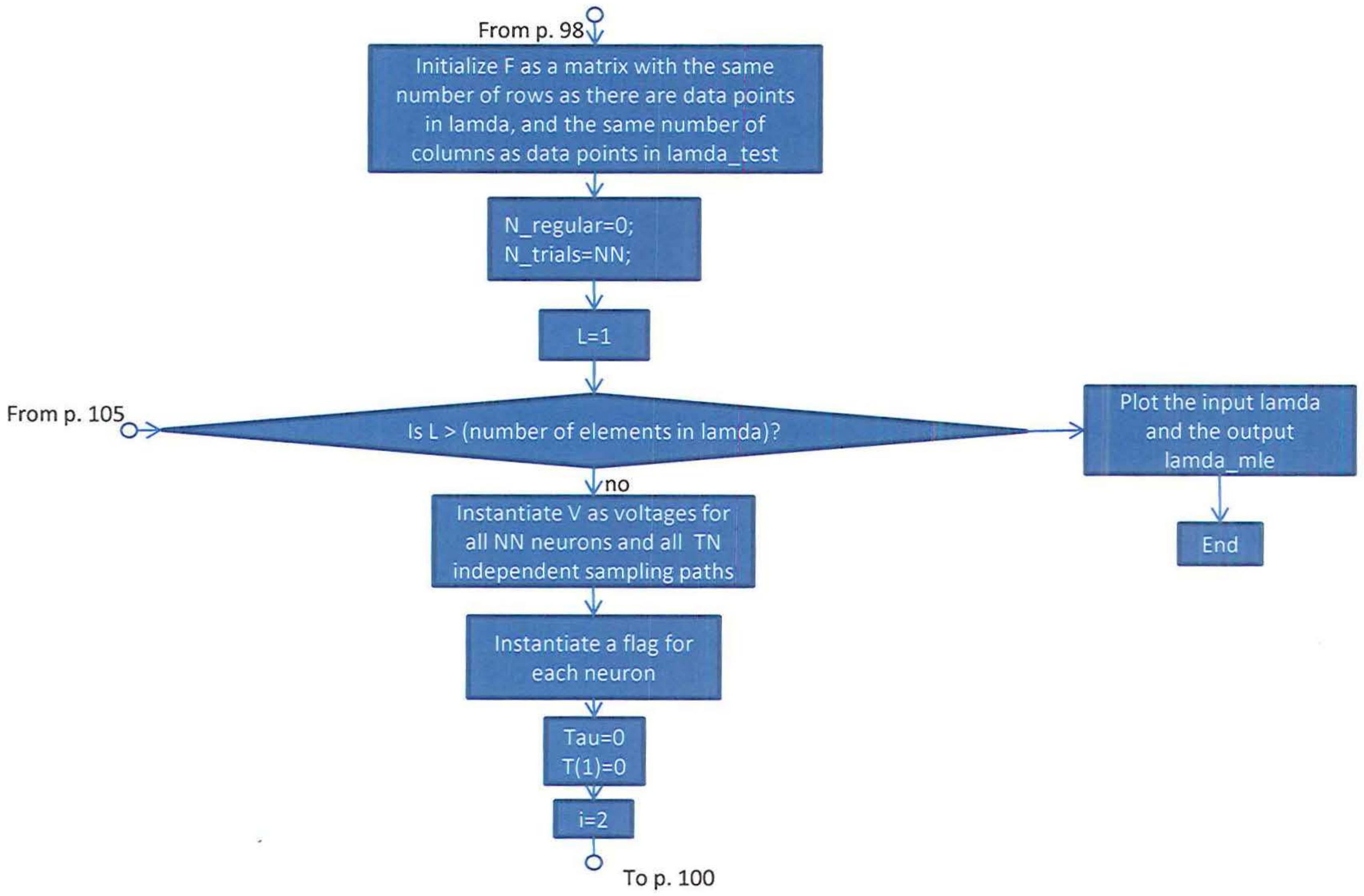
- test the BSNN's performance compared to second-generation ANNs on benchmark problems
- refine the BSNN framework to more swiftly and efficiently perform computation on continuous-valued data
- test the BSNN on the same large-scale problems as the ASNN to determine whether it has superior scaling properties to ASNNs and second-generation ANNs

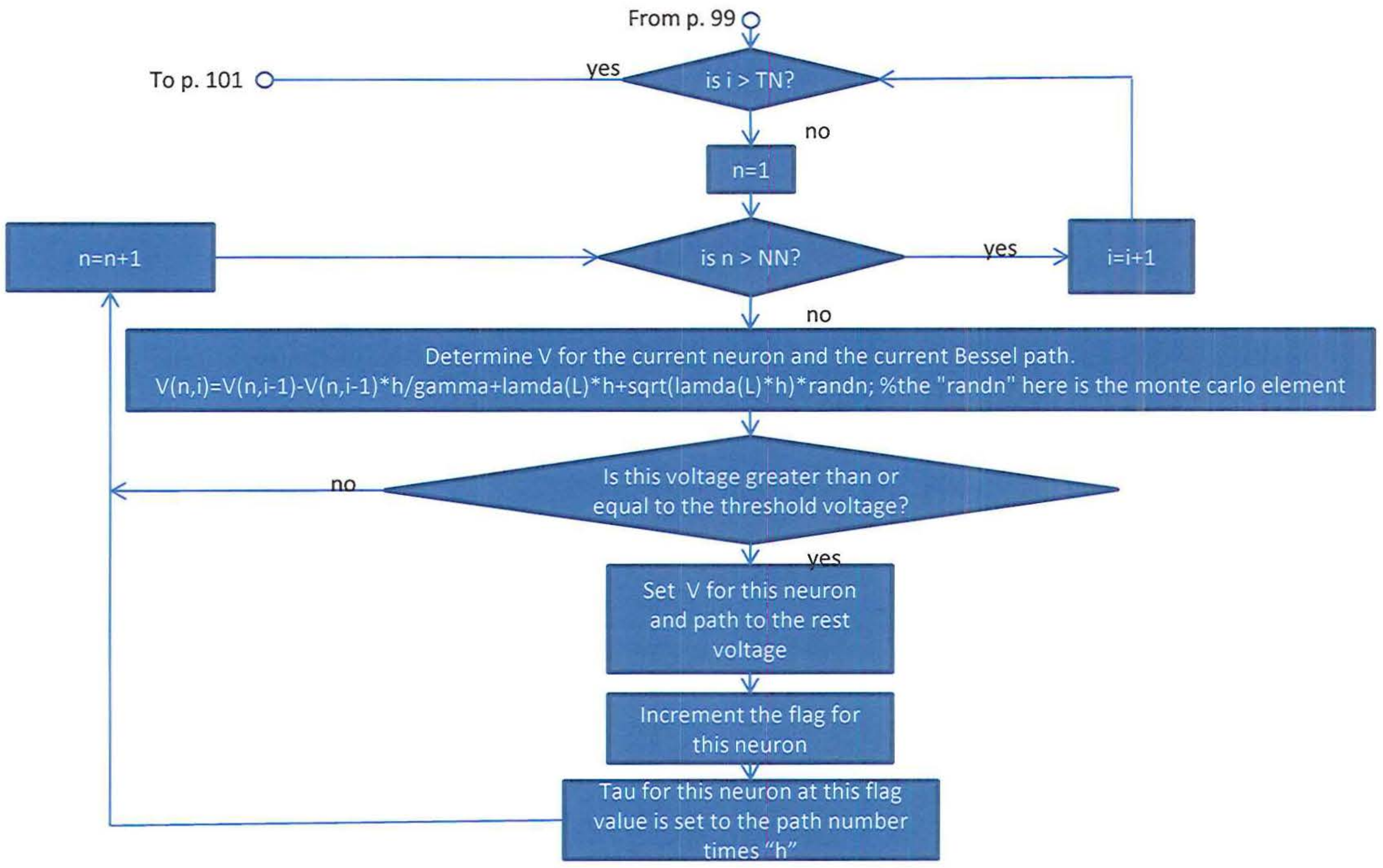
Proving that a BSNN can resolve or even invert the scaling problem would open up tremendous new vistas of computational possibilities, not the least of which would be capacity for functional modeling of enormous numbers of dynamic elements with relatively few processing units. It may not be beyond the realm of possibility for PSNs to be developed which can numerically model every particle in a physical object in reasonable time!

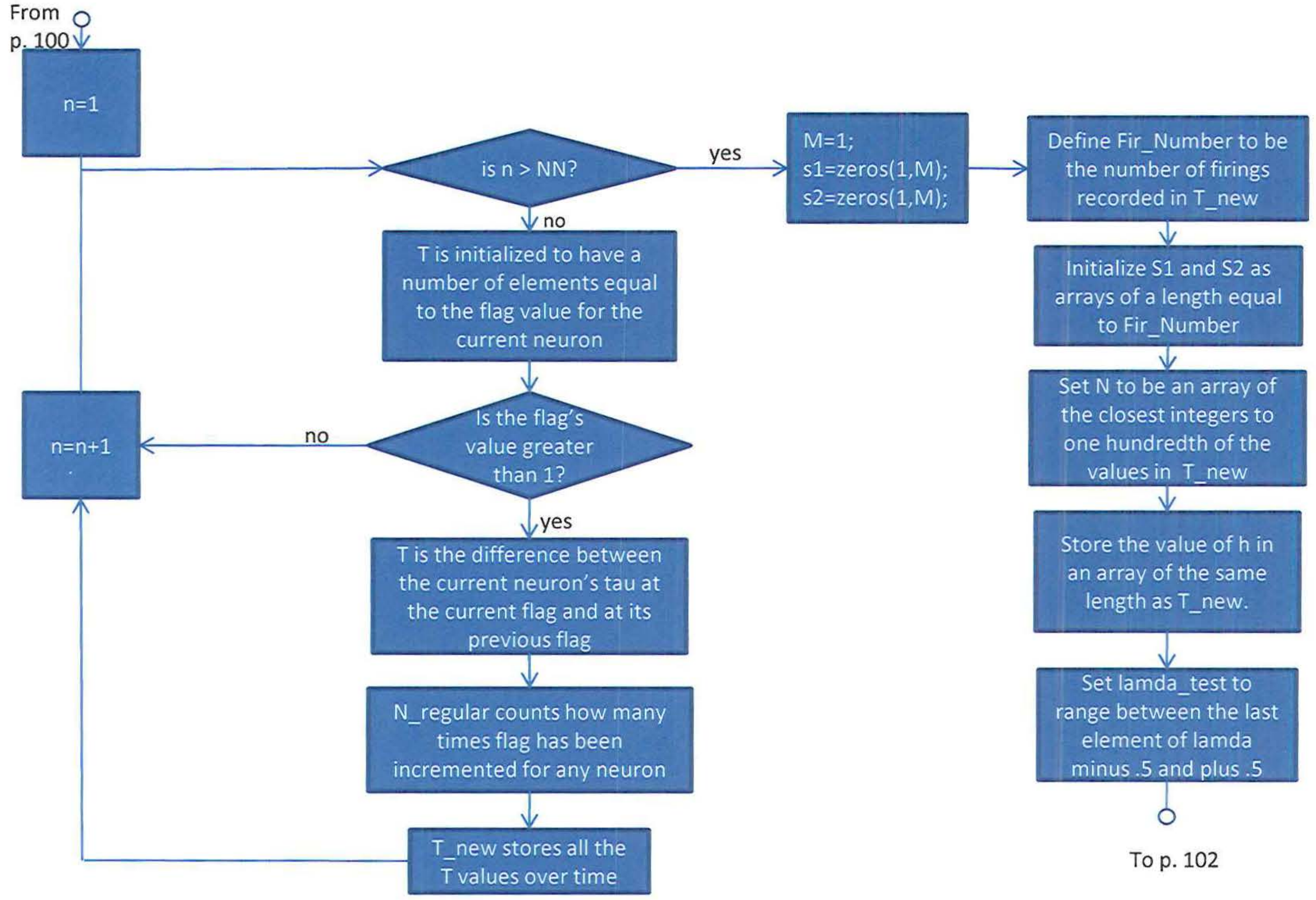
```
gamma=20;  %decay constant
V_rest=0;  %resting voltage of the LIF neuron
V_th=20;   %threshold voltage of the LIF neuron (presumably in mV)



h=0.01; % needed on first run of the code for line 41.
NN=100;%NN : 100 neurons.
TN=2000;  % "Large number" of independent sampling paths for the three-
dimensional Bessel bridge
%x=0:0.01:6;
lamda0=2;  %starting point for the input
nu=pi;     %nu is a constant in the equation
lamda_mle=0;%initialize the maximum likelihood estimate of lamda
```

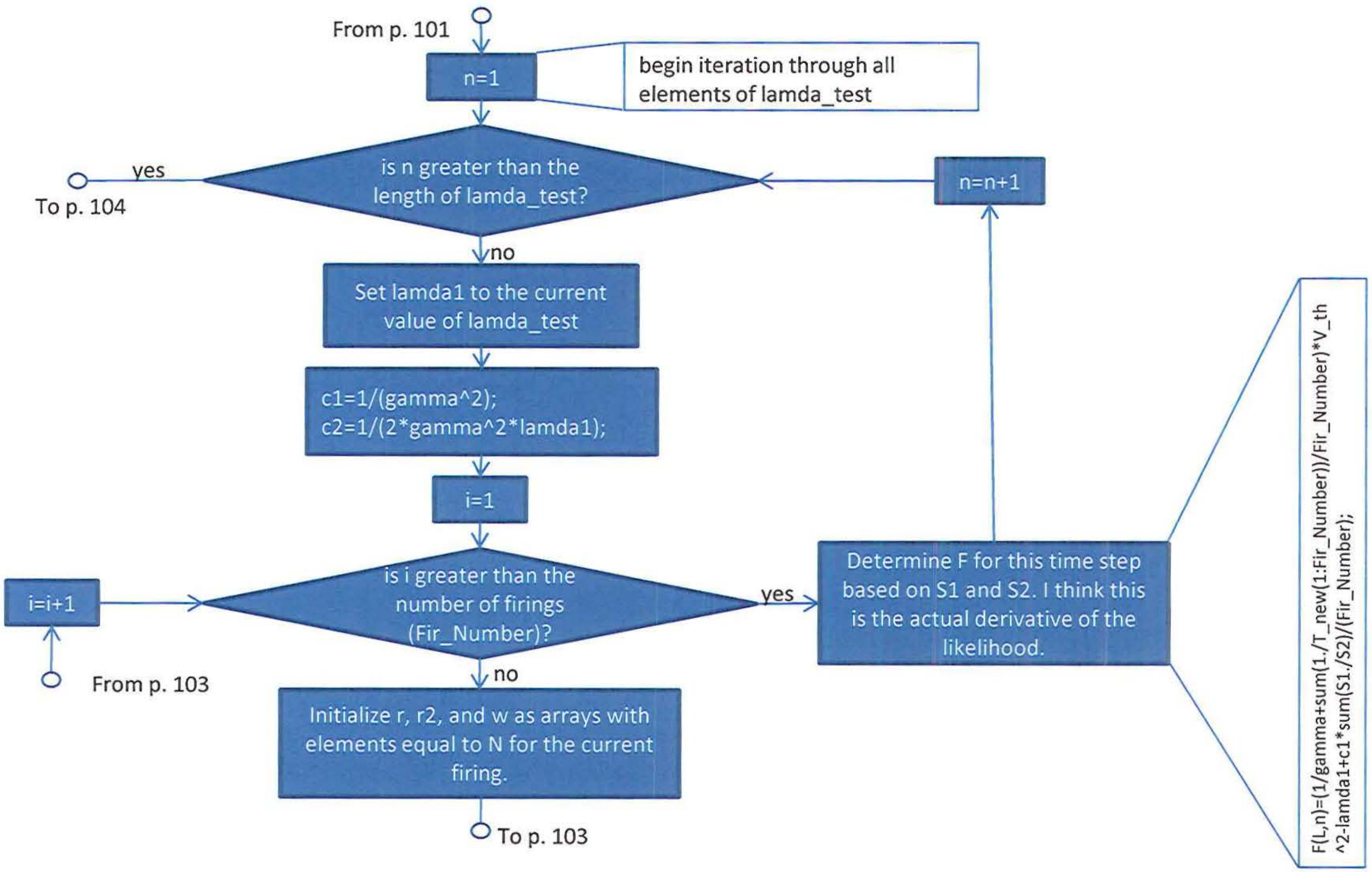
```
%    x=0.15:0.01:2; % x is the time-value for the time-varying input
rate lamda
   x=-2:.01:2;

lamda=lamda0+2*lamda0*(sin(nu*x).^2+sin(0.75*nu*x).^2);
%    lamda=2.*x.^2+300;
%    lamda = 10.*sin(x);
```

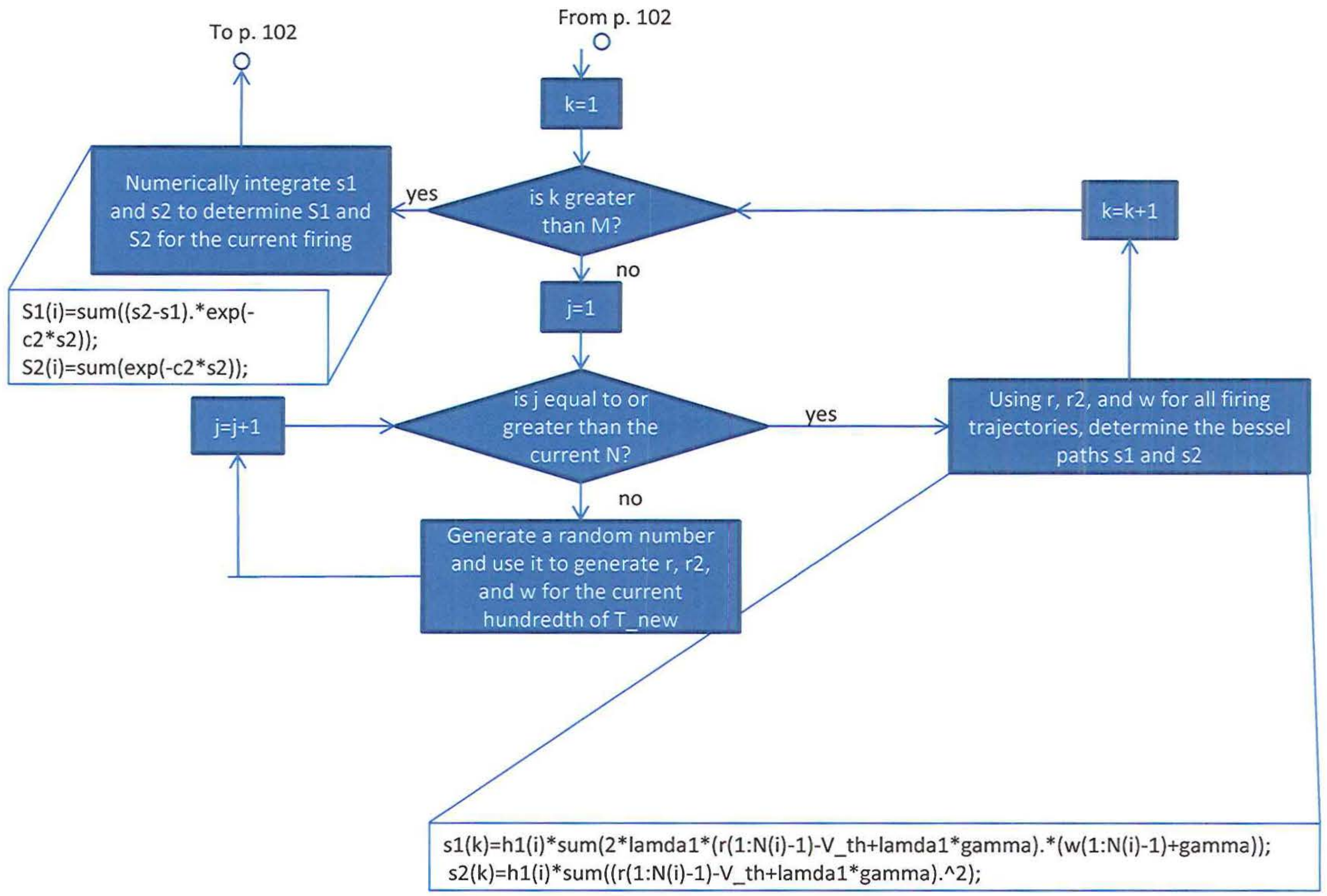

start
Assign constants for the script
m=1
Is m>1?
no
Initialize x's domain x is the independent variable in the input function
Establish lambda's range Lamda is the dependent variable of the input function, and what is actually used as the individual inputs
Set lamda_test to a range of size 1 centered on lamda(1)
To p. 99


From p. 98
Initialize F as a matrix with the same number of rows as there are data points in lamda, and the same number of columns as data points in lamda_test
N_regular=0;
N_trials=NN;
L=1
From p. 105
Is L > (number of elements in lamda)?
no
Plot the input lamda and the output lamda_mle
End
Instantiate V as voltages for all NN neurons and all TN independent sampling paths
Instantiate a flag for each neuron
Tau=0
T(1)=0
i=2
To p. 100


From p. 99
To p. 101
yes
is i > TN?
no
n=1
n=n+1
is n > NN?
yes
i=i+1
no
Determine V for the current neuron and the current Bessel path.
V(n,i)=V(n,i-1)-V(n,i-1)*h/gamma+lamda(L)*h+sqrt(lamda(L)*h)*randn; %the "randn" here is the monte carlo element
no
Is this voltage greater than or equal to the threshold voltage?
yes
Set V for this neuron and path to the rest voltage
Increment the flag for this neuron
Tau for this neuron at this flag value is set to the path number times "h"

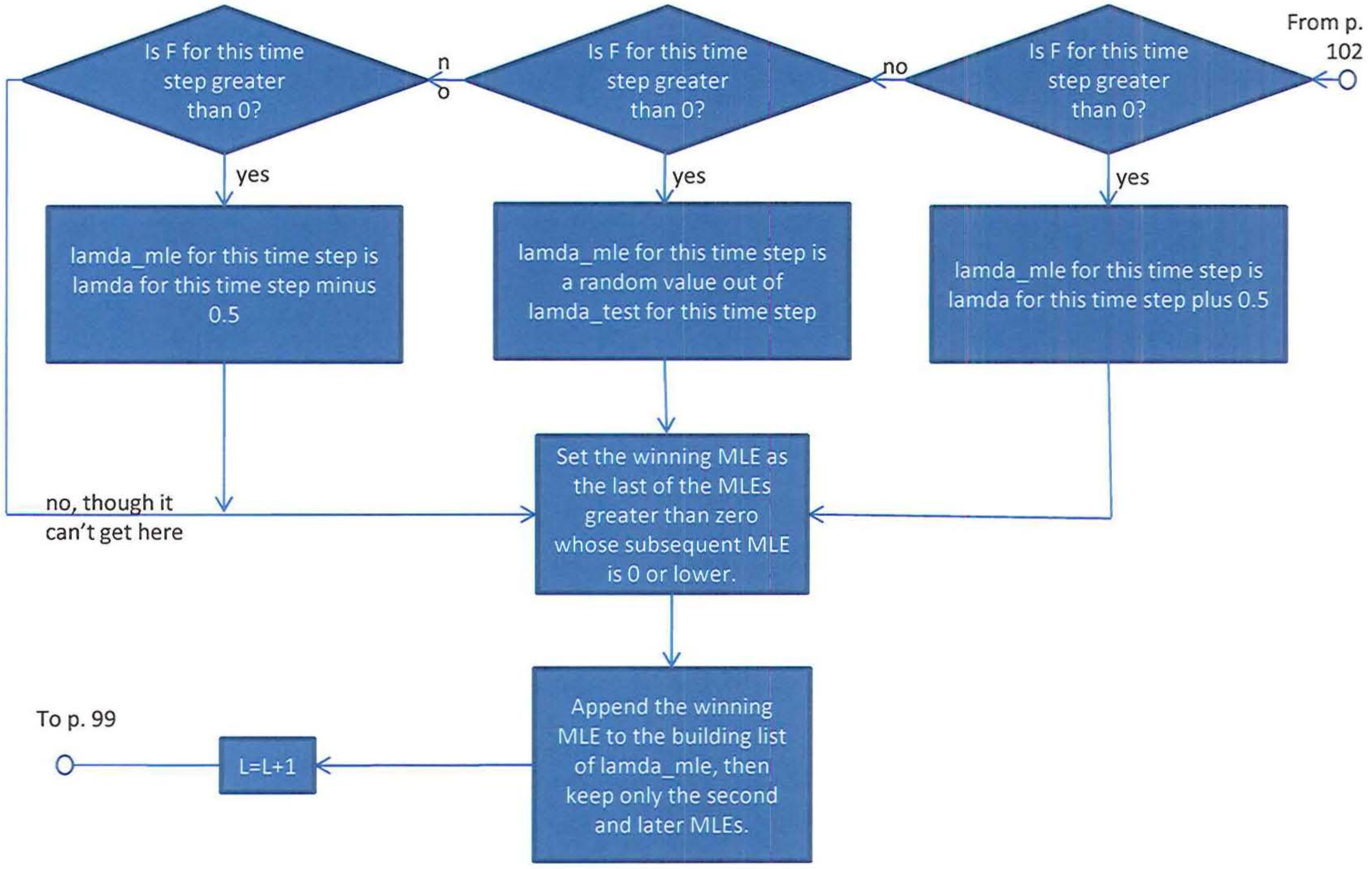

From p. 100
n=1
is n > NN?
yes
no
M=1;
s1=zeros(1,M);
s2=zeros(1,M);
Define Fir_Number to be the number of firings recorded in T_new
T is initialized to have a number of elements equal to the flag value for the current neuron
Initialize S1 and S2 as arrays of a length equal to Fir_Number
n=n+1
no
Is the flag's value greater than 1?
Set N to be an array of the closest integers to one hundredth of the values in T_new
yes
T is the difference between the current neuron's tau at the current flag and at its previous flag
Store the value of h in an array of the same length as T_new.
N_regular counts how many times flag has been incremented for any neuron
Set lamda_test to range between the last element of lamda minus .5 and plus .5
T_new stores all the T values over time
To p. 102


From p. 101
n=1
begin iteration through all elements of lamda_test
is n greater than the length of lamda_test?
yes
To p. 104
no
n=n+1
Set lamda1 to the current value of lamda_test
c1=1/(gamma^2);
c2=1/(2*gamma^2*lamda1);
i=1
is i greater than the number of firings (Fir_Number)?
yes
no
i=i+1
From p. 103
Determine F for this time step based on S1 and S2. I think this is the actual derivative of the likelihood.
F(L,n)=(1/gamma+sum(1./T_new(1:Fir_Number))/Fir_Number)*V_th^2-lamda1+c1*sum(S1./S2)/(Fir_Number);
Initialize r, r2, and w as arrays with elements equal to N for the current firing.
To p. 103


To p. 102
From p. 102
k=1
is k greater than M?
yes
no
Numerically integrate s1 and s2 to determine S1 and S2 for the current firing
k=k+1
j=1
is j equal to or greater than the current N?
yes
no
j=j+1
Using r, r2, and w for all firing trajectories, determine the bessel paths s1 and s2
Generate a random number and use it to generate r, r2, and w for the current hundredth of T_new

```
S1(i)=sum((s2-s1).*exp(-
c2*s2));
S2(i)=sum(exp(-c2*s2));
```

```
s1(k)=h1(i)*sum(2*lamda1*(r(1:N(i)-1)-V_th+lamda1*gamma).*(w(1:N(i)-1)+gamma));
 s2(k)=h1(i)*sum((r(1:N(i)-1)-V_th+lamda1*gamma).^2);
```


From p. 102
Is F for this time step greater than 0?
no
yes
lamda_mle for this time step is lamda for this time step plus 0.5
Is F for this time step greater than 0?
no
yes
lamda_mle for this time step is a random value out of lamda_test for this time step
Is F for this time step greater than 0?
yes
lamda_mle for this time step is lamda for this time step minus 0.5
no, though it can't get here
Set the winning MLE as the last of the MLEs greater than zero whose subsequent MLE is 0 or lower.
Append the winning MLE to the building list of lamda_mle, then keep only the second and later MLEs.
L=L+1
To p. 99

BIBLIOGRAPHY

[1] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron

[2] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Journal Physiol,* Vol 117, pp. 500-544, 1952.

[3] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural Networks*, Vol. 10, Issue 9, pp. 1659-1671, December 1997.

[4] P. Werbos, "Beyond regression: new tools for prediction and analysis in the behavior sciences," Ph.D. thesis, Harvard University, 1974.

[5] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE,* Vol. 78, No. 10, pp. 1550-1560, October 1990.

[6] W. Maass, G. Schnitger, and E. Sontag, "On the computational power of sigmoid versus Boolean threshold circuits," *Proceedings of the 32nd Annual IEEE Symposium on Foundations of computer Science*, pp. 767-776, 1991.

[7] D. B. Thomas and W. Luk, "FPGA Accelerated Simulation of biologically Plausible Spiking Neural Networks," *17th IEEE Symposium on Field Programmable Custom Computing Machines,* pp. 45-52, 2009.

[8] L.-C. Caron, F. Mailhot, and J. Rouat, "FPGA implementation of a spiking neural network for pattern matching," *IEEE International Symposium on Circuits and Systems*, pp. 649-652, 2011.

[9] S. Schliebs, A. Mohemmed, and N. Kasabov, "Are probabilistic spiking neural networks suitable for reservoir computing?" *International Joint Conference on Neural Networks*, pp.3156-3163, 2011.

[10] S. Ghosh-Dastidar and H. Adeli, "Improved spiking neural networks for EEG classification and epilepsy and seizure detection," *Integrated Computer-Aided Engineering*, Vol. 14, Issue 3, August 2007.

[11] S. M. Bohte, J. N. Kok, and H. LaPoutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, Vol. 48, Issues 1-4, pp. 17-37, October 2002.

[12] Q.X. Wu, T.M. McGinnity, L.P. Maguire, B. Glackin, and A. Belatreche, "Learning under weight constraints in networks of temporal encoding spiking neurons," *Neurocomputing*, Vol. 69, pp. 1912-1922, August 2006.

[13] S. G. Wysoski, L. Benuskova, and Nikola Kasabov, "Fast and adaptive network of spiking neurons for multi-view visual pattern recognition," *Neurocomputing,* Vol. 71, Issues 13-15, pp. 2563-2575, August 2008.

[14] C. Godin, M. B. Gordon, and J.D. Muller, "SpikeCell: a deterministic spiking neuron," *Neural Networks,* Vol. 15, Issue 7, pp. 873-879, September 2002.

[15] M. A. Bhuiyan, R. Jalasutram, and T. M. Taha, "Character recognition with two spiking neural network models on multicore architectures," *IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing,* pp. 29-34, 2009.

[16] S. Ratanasingam and T. M. McGinnity, "A comparison of encoding schemes for haptic object recognition using a biologically plausible spiking neural network," *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pp. 3446-3453, 2011.

[17] S.-Y. Fu, G-.S. Yang, and Z.-G. Hou, "Spiking neural networks based cortex like mechanism: A case study for facial expression recognition," *International Joint Conference on Neural Networks*, pp. 1637-1642, 2011.

[18] C. Näger, J. Storck, and G. Deco, "Speech recognition with spiking neurons and dynamic synapses: a model motivated by the human auditory pathway," *Neurocomputing*, Vol. 44-46, Issues 1-2, pp. 937-942, June 2002.

[19] S. Soltic, S. G. Wysoski, and N. K. Kasabov, "Evolving spiking neural networks for taste recognition," *IEEE International Joint Conference on Neural Networks*, IJCNN 2008, pp. 2091-2097, 2008.

[20] B. DasGupta and G. Schnitger, "The power of approximating: a comparison of activation functions," Morgan Kaufman, *Advances in neural information processing systems,* Vol. 5, pp. 615-622, San Mateo, 1993.

[21] B. Luitel and G. K. Venayagamoorthy, "Wide area monitoring in power systems using cellular neural networks," *IEEE Symposium Series on Computational Intelligence (SSCI),* CIASG, April 11-15, 2011.

[22] N. Iannella and A. D. Back, "A spiking neural network architecture for nonlinear function approximation," *Neural Networks,* pp. 933-939, 2001.

[23] W. Mass, "Fast sigmoidal networks via spiking neurons", *Neural Computation*, Vol. 9, pp. 279-304, 1997.

[24] J. U. Duncombe, "In spiking neural network architecture for nonlinear function approximation", *Neural Networks*, Vol. 14, Issues 6-7, pp. 933-939, July 2001.

[25] P. Rowcliffe and J. Feng, "Training spiking neuronal networks with applications in engineering tasks," *IEEE Transactions on Neural Networks,* Vol. 19, No. 9, pp. 1626-1640, 2008.

[26] V. Sharma and D. Srinivasan, "A spiking neural network based on temporal encoding for electricity price time series forecasting in deregulated markets," *The 2010 International Joint conference on Neural Networks*, pp. 1-8, 2010.

[27] X. Zhang, G. You, T. Chen, and J. Feng, "Maximum likelihood decoding of neuronal inputs from an interspike interval distribution," *Neural Computation,* Vol. 21, pp. 3079-3105, 2009.

[28] S. M. Bohte, H. La Poutre, and J. N. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks," *IEEE Transactions on Neural Networks,* Vol. 13, No. 2, pp. 1-10, March 2002.

[29] C. Johnson, G. K. Venayagamoorthy, "Encoding values into polychronous spiking networks," *International Joint Conference on Neural Networks,* pp. 1800-1806, 2010.

[30] C. Johnson, S. Roychowdhury, and G. K. Venayagamoorthy, "A reversibility analysis of encoding methods for spiking neural networks," *Proceedings of IJCNN 2011.*

[31] C. Johnson, G. K. Venayagamoorthy, and P. Mitra, "Comparison of a spiking neural network and an MLP for robust identification of generator dynamics in a multimachine power system," *Neural Networks,* Vol. 22, pp. 833-841, 2009.

[32] A. P. Engelbrecht, *Computational Intelligence: an introduction,* Wiley, 2002.

[33] L. O. Chua and L. Yang, "Cellular neural networks: theory," *IEEE Transactions on Circuits and Systems,* Vol. 35, pp. 1257-1272, 1988

[34] J. Dethier, V. Gilja, and P. Nuyujukian, "Spiking neural network decoder for brain-machine interfaces," *5th International IEEE/EMBS Conference on Neural Engineering,* pp. 396-399, 2011.

[35] N. Kasabov, L. Benuskova, and S. G. Wysoski, "A computational neurogenetic model of a spiking neuron," *2005 IEEE International Joint Conference on Neural Networks,* IJCNN '05, Vol. 1, pp. 446-451, 2005.

[36] D. A. Wagenaar, J. Pine, S. M. Potter, "Effective parameters for stimulation of dissociated cultures using multi-electrode arrays," *Journal of Neurscience Methods,* Vol. 138, pp. 27-37, 2004.

[37] http://www.neuro.gatech.edu/groups/potter/ Laboratory for Neuroengineering, September 2011.

[38] R. Ortman, G. K. Venayagamoorthy, and S. Potter, "Input separability in living liquid state machines," 10th International Conference on Adaptive and Natural Computing Algorithms, Ljubljana, Slovenia, April 2011.

[39] H. Torikai, T. Saito, "Analysis of various spike-trains from a digital spiking neuron," *Inernational Congress Series,* Vol. 1291, pp. 225-228, June 2006.

[40] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks,* Vol. 14, pp. 1569-1572, 2003.

[41] W. Zhang, Q. Qiao, and X. Zheng, "Associative memory and segmentation in a network composed of Izhikevich neurons," Fourth International Conference on Natural Computation, pp. 618-621, 2008.

[42] J. M. Nageswaran, N. Dutt, J.L. Krichmar, A. Nicolau, and A. Veidenbaum, "Efficient simulation of large-scale spiking neural networks using CUDA graphics processors," International Joint Conference on Neural Networks, pp. 2145-2152, 2009.

[43] E. Izhikevich, "Polychronization: computation with spikes," *Neural Computation*, Vol. 18, pp. 245-282, 2006.

[44] M. Molina, J. Liang, R. Harley, and G. K. Venayagamoorthy, "Comparison of TDNN and RNN performances for neuro-identification of small to medium-sized power systems," *IEEE Symposium Series on Computational Intelligence (SSCI)*, April 11-15, 2011.

[45] S. M. Ross, "The Poisson Process," in *Stochastic Processes*, Chapter 2. Wiley, pp. 59-97, 1995. ISBN 0-13-711564-4.

[46] E. M. Izhikevich, J. A. Gally, and G. M. Edelman, "Spike-Timing Dynamics of Neuronal Groups," *Cerebral Cortex*, Vol. 14, pp. 933-944, 2004.

[47] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," Proceedings of the IEEE International Conference on Neural Networks, pp. 1942-1948, 1995.

[48] G. K. Venayagamoorthy, "Online design of an echo state network based wide area monitor for a multi-machine power system," *Neural Networks*, Vol. 20, Issue 3, pp. 404-413, 2007.

[49] M. Azmy, I. Erlich, and P. Sowa, "Artificial neural network-based dynamic equivalents for distribution systems containing active sources," *IET Proceedings on Generation, Transmission Distribution,* Vol. 151, Issue 6, pp. 681-688, 2004.

[50] M. Stankovic, A. T. Sarik, and M. Milosevic, "Identification of nonparametric dynamic power system equivalents with artificial neural networks," *IEEE Transactions on Power System,* Vol. 18, Issue 4, pp. 1478-1486, 2003.

[51] J.-W. Park, G. K. Venayagamoorthy, and R. G. Harley, "MLP/RBF neural networks based on-line global model identification of synchronous generator," *IEEE Transactions on Industrial Electronics,* Vol. 52, Issue 6, pp. 1685-1695, 2005.

[52] S. Singh and G. K. Venayagamoorthy, "Online identification of turbogenerators in a multimachine power system using RBF neural networks," *Artificial neural networks in engineering conference*, pp. 485-490, 2002.

VITA

Cameron Eric Johnson has studied at the University of Missouri at Rolla since Fall of 2000, when he was a freshman in the Physics department. He earned his B.S. in Physics in 2004, went on to receive a M.S. in Physics in 2006 and, after determining that he wished to develop intelligent systems to control autonomous swarms of devices, earned an M.S. in Computer Engineering in 2007. He worked towards his Ph.D. as a GAANN fellow. He received his Ph.D. in Computer Engineering with the completion of this dissertation in the Fall of 2011.

His dissertation is on spiking neural networks, which represents the focus of his research and breakthroughs in computational intelligence, which remains his broader area of interest.

Cameron is interested in developing advanced devices and systems which automate the mental work of experts in the way that robots in factories automated the physical labor of unskilled workers, enabling one expert to do the work of dozens. He is also interested in developing smarter and more intuitive systems to improve the productivity and capabilities of end users in day-to-day life.