MODELING OF ELECTRICAL CIRCUIT WITH RECURRENT NEURAL NETWORKS

BY

ZAICHEN CHEN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

ABSTRACT

In this dissertation, a circuit modeling methodology using recurrent neural networks (RNNs) is developed. The methodology covers model structure selection, data generation, training, and model implementation for circuit simulation. Several different RNN structures are investigated and their capabilities in circuit modeling are compared. The stability of RNN in the context of circuit modeling is defined and methods to guarantee stability for some RNN structures are developed. The modeling methodology is supported by test cases showing the accuracy and efficiency of RNN models.

ACKNOWLEDGMENTS

I would like to give sincere thanks to my advisor, Professor Elyse Rosenbaum, for her kind encouragement and guidance throughout my graduate study. I would also like to thank Professor Maxim Raginsky for his help during my doctoral research. Finally, I would like to thank former and current fellow students, including Dr. Kuo-Hsuan Meng, Dr. Robert Mertens, Dr. Min-Sun Keel, Dr. Nicholas Thomson, Dr. Yang Xiu, Collin Reiman, Sandeep Vora, Jie Xiong, and Xiao Ma, for their help and support during my life in Illinois.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Modeling of electrical circuit components has always been a critical enabler for electronic design automation (EDA). In general, models used in circuit simulation can be classified into one of two major categories. For semiconductor devices, physics-based models can be derived from an analysis of the device physics, e.g. electrostatics, carrier transport, generation and recombination, etc. Those models are usually accurate and robust in various simulation conditions. However, the development cost of physics-based models is high in both time and labor, and they are usually computationally inefficient especially for large-scale circuits. In addition, for deep-submicron semiconductor devices, the underlying physics becomes too complicated to model with simple analytical equations. Consequently, accurate descriptions of the device physics are replaced by approximations with empirical fitting parameters introduced, compromising the model accuracy. On the other hand, behavioral models can be developed for either a single semiconductor device, or a circuit block consisting of many semiconductor devices and/or other components. The model equations of a behavioral model are arbitrarily chosen to best represent the observed characteristics of the entity being modeled. Compared with physics-based models, behavioral models are often more computationally efficient and less costly to develop.

While physics-based models and behavioral models are both used in the integrated circuit (IC) design process, the situation changes when the design process of an electrical system, e.g. cell phone, laptop, etc., is considered. In principle, it is possible to represent an entire electrical system with a netlist of components, and run simulation with all components described by physics-based models. Practically, however, IC designers need to protect their intellectual property (IP), i.e. their design; thus the circuit netlist cannot be provided to system designers. Without a netlist, physics-based models cannot be used. In addition, even when the circuit netlist is available, simulation of a full electrical system with physics-based models often turns out to be too time consuming. Therefore, IC modeling in system design is usually achieved through behavioral models. For instance, IBIS is an industry standard modeling template for IC packages and I/O circuits in which the IC terminal current-voltage (I-V) characteristics are modeled with lookup tables and fixed waveforms.

In this work, it is proposed to use the recurrent neural network (RNN) as a behavioral model of semiconductor devices and electrical circuit blocks. Throughout this dissertation, the physical entity (device or circuit block) being modeled will be referred to as "circuit being modeled," "original circuit," or simply "circuit," even if it is only a single semiconductor device. The RNN is a black-box model since the model equations are not designed to match any specific system; rather, the model equations are designed for versatility and ease of training. Note that the word "training" is used to describe the optimization of model parameters against known data, and those data will be referred to as "training data." The black-box nature of RNN makes it inherently IP-obscuring, and its versatility makes it viable for various types of circuits, reducing the need to develop multiple model structures and parameter extraction techniques for different classes of circuits. Among other neural network models that share the black-box property, RNN is chosen since it is proven [1] to model nonlinear systems that can be described by nonlinear state-space equations, specifically

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \tag{1.1a}$$

$$\boldsymbol{y} = g(\boldsymbol{x}, \boldsymbol{u}) \tag{1.1b}$$

where $\boldsymbol{u}$ is the vector of inputs, $\boldsymbol{y}$ is the vector of outputs, and $\boldsymbol{x}$ is the vector of the internal states of the system. Physics-based models of semiconductor devices can in general be written in the form of nonlinear state-space equations (1.1). Thus, it is expected that RNN can model circuits that consist of mostly semiconductor devices. The suitability of RNN for the modeling of general dynamic systems is also demonstrated in [2].

Potentially, RNN is especially useful to model electrostatic discharge (ESD) protection circuits. The requirement for electrical systems to pass system-level ESD tests, e.g. the IEC 61000-4-2 standard [3], raises the need for ESD-accurate IC modeling. The current industry standard of IC modeling, the IBIS model, only includes a static I-V lookup table for ESD purposes and cannot accurately model the transient response of the on-chip ESD circuit. An alternative method proposed by the Industry Council — System-Efficient ESD Design (SEED) — attempts to extract an on-chip ESD circuit model through pulsed I-V measurements of the IC. As a result, SEED also fails to address the transient behavior of on-chip ESD circuit [4]. Using physics-based ESD device models in conjunction with the IBIS model may accurately predict the IC's transient behavior, but the parameter fitting process for ESD device models can be difficult due to limited measurement capabilities for devices operating at ESD power levels. Instead, RNN may provide a generalized behavioral modeling method for ESD protection circuits. An RNN based on-chip ESD circuit model captures both the transient and static behavior of the circuit. The RNN model can be derived by the IC designer using training data generated from circuit simulation with the full netlist, and distributed to system designers for accurate transient ESD simulation without IP disclosure.

1.2 Review of Previous Work

The problem of deriving behavioral models for unknown nonlinear systems is usually referred to as nonlinear system identification. For the general problem of nonlinear system identification, several classes of black-box models have been proposed, including but not limited to the Volterra series, the Hammerstein-Wiener model, autoregressive models, neural networks, and models in some general nonlinear state-space form. Obviously, the problem of circuit modeling can be considered a nonlinear system identification problem, thus all these black-box models can potentially be applied. Indeed, there have been numerous publications discussing the use of general black-box models for circuit modeling. In this section, a comprehensive literature review is presented focused on those publications, and the novelty of this work compared with the past publications is explained.

The Volterra series [5] has long been used for modeling of electronic devices and circuits. In [6, 7], Volterra series is used to model a MESFET device, and in [8] an electrodynamic speaker. Many prior works use the Volterra series for the modeling of power amplifiers, such as in [9, 10, 11, 12, 13, 14]. Some variations or improved versions of the Volterra series are used in those works; for instance, the truncated Volterra series is used in [12], and a model order reduction method is used in conjunction with the Volterra series in [14].

The Hammerstein-Wiener model consists of cascaded blocks of linear transfer functions and static nonlinear mappings. Models of this class are used to model power amplifiers in [15, 16], wireless transmitters in [17], DC/DC converters in [18], and IGBTs in [19]. In this work, the Hammerstein-Wiener model was also initially chosen as a candidate model for ESD protection circuits. However, it turned out that for the types of nonlinearity exhibited by ESD protection circuits, the Hammerstein-Wiener model has difficulty fitting to the data.

Autoregressive models use discrete-time functions that depend on past information of the system input and output. At each time step, the output of the system is calculated from a nonlinear static function whose inputs consist of the current system input, the system input from several previous time steps, and the system output from several previous time steps. Some popular choices of the nonlinear static function include polynomials, radial basis functions (RBF), and feedforward neural networks (FNN). Usually, the acronym "NARX model" stands for nonlinear autoregressive models, where the letter X stands for eXogenous (i.e. external) input. In [20], NARX model with RBF nonlinearity is used to model RF circuits. In [21, 22, 23, 24, 25, 26], NARX model with FNN nonlinearity is used to model RF power amplifiers, and the same model structure is used in [27] for RF receivers, and in [28] for a variety of electrical circuits. Among those publications, some improvements to the modeling procedure are also proposed. For example, the stability of the NARX model with a specific FNN nonlinearity is discussed in [22]; an optimal training data generation method is proposed in [23]; and a novel model structure in which the inputs to the nonlinear static function are connected to multiple layers of an FNN is introduced in [25]. Instead of using a pre-determined analytical nonlinear function, techniques have also been developed to extract the nonlinearity in the NARX model from data, such as in [29, 30].

The FNN is also used in circuit modeling as non-autoregressive models. In [31], FNN is used to describe the input-output relationship of the magnitude and phase components of a power amplifier. In [32, 33], FNN is used for frequency-domain modeling of RF circuits. In [34], a state-space equation for an oscillator is proposed with FNN as part of the equation.

In this work, circuit modeling technique is developed for RNN structures, which are commonly used for applications such as natural language processing [35] and handwriting recognition [36]. The RNN model structure is fundamentally different from the Volterra series or Hammerstein-Wiener model, as the latter two do not contain any form of feedback. The NARX models with FNN nonlinearity have a structure similar to that of the RNNs investigated in this work, except that NARX models use feedback from the output while RNN models use feedback from a hidden layer in the network. To the best of the author's knowledge, this is the first such RNN structure to be investigated for circuit modeling purposes. In addition, most of the previous work focuses on RF circuits such as power amplifiers, and the models are designed to be used for simulation with constant time steps. In contrast, this dissertation investigates the use of RNN model structures for general purpose circuit modeling. In this work, the simulation environment is expected to be a general-purpose circuit simulator, which will usually adopt a variable time step in transient simulation. Finally, unlike most previous work, this dissertation systematically analyzes the capacity of RNN model structures, signifying both their advantages and weaknesses.

CHAPTER 2

GENERAL RNN CIRCUIT MODELING METHODOLOGY

All circuit models, physics-based or behavioral, consist of the model structure and the model parameters. The model structure is a set of mathematical equations appropriate for describing a class of circuit. Each specific circuit in the class can be represented by those equations by adjusting the model parameters accordingly. Thus, modeling a circuit consists of three steps:

1. Choose a model structure based on the circuit class.

2. Collect data for the specific circuit.

3. Fit model parameters to the data.

This process is applicable for both physics-based and behavioral modeling. In this chapter, an introduction to the RNN circuit modeling methodology is given in these three steps.

2.1 General RNN Model Structure

2.1.1 General RNN Model Equations

Consider a general nonlinear system with n_u inputs and n_y outputs. Denoting the system input $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ and output $\boldsymbol{y}(t) \in \mathbb{R}^{n_y}$, the RNN model of the system can be expressed in the following general form:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), \Theta_x) \tag{2.1a}$$

$$\boldsymbol{y}(t) = g(\boldsymbol{x}(t), \Theta_y) \tag{2.1b}$$

Here, $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ is the hidden state vector and its dimension n_x is an empirically selected model parameter. $\dot{\boldsymbol{x}}(t) = \frac{d\boldsymbol{x}(t)}{dt}$ is the time derivative of the hidden states, f and g are static functions, Θ_x and Θ_y together comprise the trainable parameters $\Theta = \Theta_x \cup \Theta_y$. For an RNN, f is a function generally considered to be an artificial neural network; its exact formulations will be introduced in Chapter 3. The function g is chosen to be affine functions of $\boldsymbol{x}$, i.e.

$$g(\boldsymbol{x}(t), \Theta_y) = \boldsymbol{b}_y + W_y \boldsymbol{x}(t)$$

where the model parameter $\Theta_y = \{\boldsymbol{b}_y, W_y\}$, $\boldsymbol{b}_y \in \mathbb{R}^{n_y}$ and $W_y \in \mathbb{R}^{n_y \times n_x}$.

For the rest of this dissertation, the explicit time dependence of the variables $\boldsymbol{u}$, $\boldsymbol{x}$, and $\boldsymbol{y}$ will be omitted in equations in which the time dependence is obvious.

2.1.2 Associate RNN Model with Circuit

In Section 2.1.1, the RNN is introduced as a state-space model which predicts a time-domain output waveform given an input waveform. For an electrical circuit model, the behavior of the circuit needs to be described in terms of its terminal voltages and currents. The correspondence between RNN inputs/outputs and terminal voltages/currents of a circuit is discussed in this section.

Consider a circuit with q external terminals. A model of the circuit must describe the relationship between all terminal voltages and currents. However, voltage is only meaningful in electrical circuits as a difference between two nodes.

Thus, one terminal of the circuit should be chosen as a reference, and all other terminal voltages should be described by the difference between them and the reference. Also, due to Kirchhoff's current law, the current flowing into the reference is always equal to the sum of current flowing out of all other terminals. As a result, there is no need to introduce the current at the reference into the model equations. Following this analysis, the circuit can be viewed as a p-port where $p = q - 1$, and each port consists of a non-reference terminal as the positive node, and the reference as the negative node. A complete circuit model can be made by equations describing the relationship between all p port voltages and p port currents.

Depending on the type of the model and the application in which it is used, the port voltages and currents of the circuit can be represented as scalar variables, time domain waveforms, frequency domain signals, etc. The RNN model developed in this work is intended mostly for transient simulations. Therefore, the port voltages and currents of the circuit are represented as time-domain waveforms. To fully describe the transient characteristics of the p-port, a behavioral model needs to enable the prediction of all port currents given all port voltages, and vice versa. To achieve this, the RNN input and output dimensions need to be chosen as $n_u = n_y = p$. Then, each port of the circuit should be associated with a single input and a single output of the RNN. Two configurations can be chosen for this association — voltage waveform as RNN input, current waveform as RNN output, or the reverse. In this way, the prediction of all port currents given all port voltages (or vice versa) becomes the problem of solving (2.1) with p known voltage and current waveforms, and p unknown voltage and current waveforms.

For certain ports of some circuits, there is no need to include both the port voltage and current in the model equations. Rather, those ports can be considered input-only or output-only. Input-only ports with voltage as the only variable can be defined for ports with such high input impedances that the voltage waveform at the port is practically independent of the current that flows into it. Some examples include gate terminals of MOSFETs driven by sufficiently large sources, or control terminals of a voltage-controlled relay. Note that for the practical implementation of a voltage input port, no branch current is associated with the port, making it effectively an open circuit. On the other hand, an input-only port with current as its variable is not compatible with circuit simulators, since the inclusion of such port requires the node voltage at the port to be undefined. This requirement violates the principle of modified nodal analysis. Output-only ports can be designated if the port will only be connected to some known fixed load. For instance, if a port is always connected to circuits with 50 Ω input resistance, one can connect the port to an actual 50 Ω resistor during training data generation, and treat the port as output-only with either voltage or current as the variable. Note that during circuit simulation, if an output-only port is connected to load that is different from what was assumed, the simulation result may deviate drastically from the behavior of the original circuit.

In summary, a circuit port can be associated with an RNN model in the following ways:

- voltage-input, current-output (VICO)
- current-input, voltage-output (CIVO)
- voltage-input, no output (VINO)
- no input, voltage-output (NIVO)
- no input, current-output (NICO)

Together, the VICO and CIVO ports are referred to as input-output ports.

2.2 Training Data Generation

To enable the parameter fitting, or training step of the modeling procedure, training data need to be collected for the circuit being modeled. In general, the training data need to be voltage and current waveforms collected at circuit ports corresponding to the associated RNN inputs and outputs. The range of the training data determines the applicability of the trained model. Since RNN is a black-box model, one cannot expect the model to make correct prediction for inputs that are vastly different from everything that is included in the training data. Conversely, if the model is expected to be used for certain applications, then the training data should cover, as much as possible, the operation range of the original circuit for those applications. The operation range of a circuit can roughly be defined by imposing limits on the amplitude, frequency, and slew rate of the voltage and current waveforms applied at its ports.

For most modeling tasks addressed in this work, the netlist of the circuit being modeled is available. In those cases, the training data can be generated from transient circuit simulation. To maximize the variability of the input waveforms yet guarantee that reasonable stimuli are being applied, a training data generation method based on random piecewise linear (PWL) sources is introduced. Figure 2.1 shows the circuit netlist used for data generation. Each of the n voltage sources in Figure 2.1 has an output waveform in the following PWL formulation:

$$v_s(t) = V_{k-1} + \frac{V_k - V_{k-1}}{\tau_k}(t - \sum_{i=1}^{k-1}\tau_i),\ \sum_{i=1}^{k-1}\tau_i \le t \le \sum_{i=1}^{k}\tau_i \tag{2.2}$$

In (2.2), V_i and τ_i are random variables selected independently for all n voltage sources. V_i with $i \in \mathbb{N}$ are the values of the PWL function at both ends of each of the linear segments. The value of V_i is sampled from a uniform distribution with lower and upper bounds V_{min} and V_{max}. Similarly, τ_i with $i \in \mathbb{N}^+$ are the lengths of the linear segments, and the value of τ_i is sampled from a log uniform distribution with lower and upper bounds τ_{min} and τ_{max}. Here, the log uniform distribution is chosen to ensure that short rising or falling edge segments consist of a significant portion of the total training data. The bounds of both distributions are determined by the operational ranges of the circuit. In this way, the amplitude and the power spectrum of the stimuli are bounded within the desired range of operation of the circuit. Each voltage waveform is smoothed with a low-pass filter whose cutoff frequency is much higher than τ_{min}^{-1} to avoid numerical difficulty caused by the discontinuity in the first derivative of $v_s(t)$. The length of each transient simulation is chosen to be 5-10 times τ_{max}. The source resistances R_i in Figure 2.1 are varied between each transient simulation. For each simulation, the values of the resistances are sampled from a log uniform distribution whose range resembles reasonable source resistances the circuit will see as part of a larger system. Figure 2.2 shows a sample PWL waveform $v_s(t)$ and the corresponding voltage and current waveforms measured at the terminals of the circuit.

In some modeling tasks, the circuit netlist is unknown and the training data have to be generated from measurement of the physical circuit. In those cases, the training data are limited to those that can possibly be generated from a physical measurement setup. Those physical systems can most likely be described by a voltage source with a series impedance, although the source impedance is usually fixed for the measurement system. In Figure 2.3, several examples of source voltage waveforms with good variability are shown. All those waveforms can be realized using a transmission line pulse (TLP) test system [37] with some modifications.

With either training data generation method — simulation or measurement — the raw data collected are in the form of sampled waveforms, i.e. pairs of

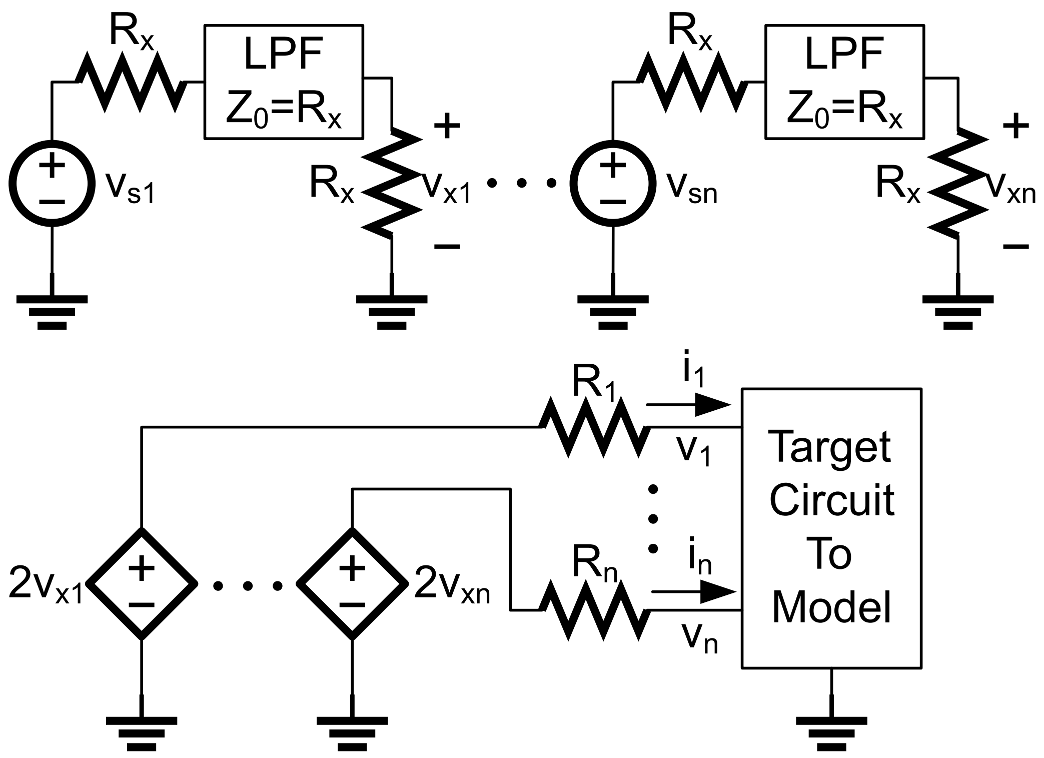


Figure 2.1: Schematic representation of the circuit used for training data generation.

time and voltage or current. The time steps in those data waveforms may not necessarily be uniform. On the other hand, the RNN structures used in this work are in general developed for training with time series data, i.e. vectors of voltage and current values with an implied uniform time step. Therefore, as a final step of training data generation, the raw data waveforms are interpolated to a time base whose time step is uniform.

2.3 Training

In machine learning, the parameter fitting step is called training. Training can be defined as the process of finding the model parameters that minimizes a loss function defined to quantify the difference between the model and the original circuit. Specifically, for a fixed set of training data, the loss function only depends

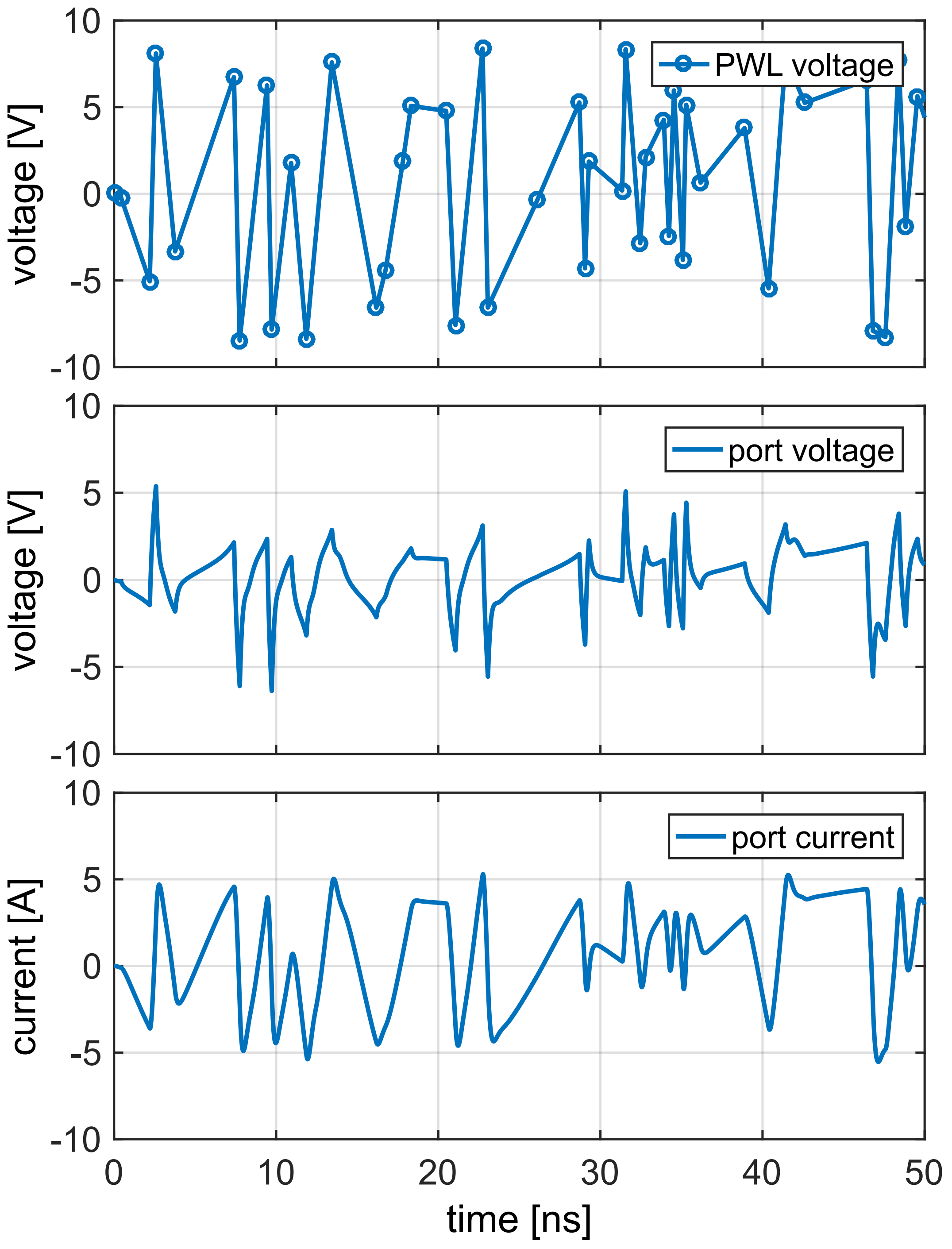


Figure 2.2: Sample random PWL training data. The voltage and current waveforms at the circuit port are not PWL functions since they are related to one another by the input impedance of the port.

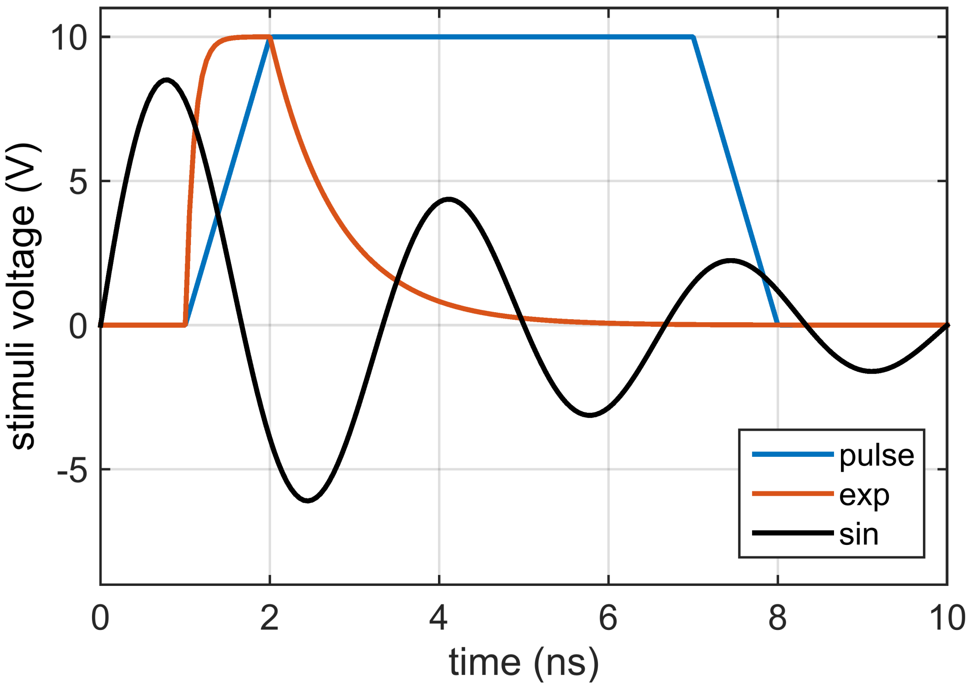


Figure 2.3: Sample selected waveform training data.

on the model parameters. Thus, general purpose optimization algorithms can be applied to minimize the loss function, and thus to obtain the best fit of the model parameters to the training data. A mathematical description of this process is given below.

Assume that the training data being provided are time series. Denote the training dataset $\{\boldsymbol{u}^k, \boldsymbol{y}^k\}$ where k labels the number of time series in the training set, the input time series is $\boldsymbol{u}^k = \{\boldsymbol{u}_1^k, \boldsymbol{u}_2^k, \boldsymbol{u}_3^k, ...\}$ and the output time series is $\boldsymbol{y}^k = \{\boldsymbol{y}_1^k, \boldsymbol{y}_2^k, \boldsymbol{y}_3^k, ...\}$. The loss function can be defined as the mean squared error (MSE) between the output of the training data and the prediction of the RNN given the input of the training data:

$$L(\boldsymbol{u}^k, \boldsymbol{y}^k, \Theta) = \sum_i \left\| \boldsymbol{y}_i^k - \hat{\boldsymbol{y}}_i(\boldsymbol{u}^k, \Theta) \right\|^2 \tag{2.3}$$

where the time series $\hat{\boldsymbol{y}}(\boldsymbol{u}^k, \Theta)$ is the prediction of the RNN given input time series $\boldsymbol{u}^k$ and parameters Θ. Then, the optimization problem can be expressed as:

$$\Theta_{best} = \arg\min_{\Theta} \sum_{k=1}^{N} L(\boldsymbol{u}^k, \boldsymbol{y}^k, \Theta) \tag{2.4}$$

It is worth noting that the MSE loss function is chosen since the expected application for the RNN model is ESD simulation. For this application, it is most important to accurately predict the amplitudes of peaks of the transient circuit response. Optimizing the MSE loss leads to good model accuracy for those peak amplitudes. However, optimization of the MSE loss does not lead to optimized model error in terms of low voltage/current response, or frequency response. Therefore, if the RNN model is expected to be used in leakage or spectral analysis, a different loss function should be used for training.

For time series training data, the RNN prediction $\hat{\boldsymbol{y}}(\boldsymbol{u}, \Theta)$ can be calculated from a discrete-time approximation of (2.1). There are multiple methods for this approximation, e.g. forward Euler, backward Euler, the trapezoidal rule, etc. Importantly, the optimization problem (2.4) is only numerically viable if $L^k(\Theta) = L(\boldsymbol{u}^k, \boldsymbol{y}^k, \Theta)$ can be expressed as an explicit function of Θ. Accordingly, it is required that a closed form expression of $\hat{\boldsymbol{y}}(\boldsymbol{u}, \Theta)$, also being an explicit function of Θ, can be derived from the discrete-time approximation of (2.1). The Forward Euler method satisfies this condition and is chosen to be the approximation method. Thus, if the training time series are created with the time step h, (2.1) is approximated with the following discrete-time recurrence equation:

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + h \cdot f(\boldsymbol{x}_{i-1}, \boldsymbol{u}_i, \Theta_x) \tag{2.5a}$$

$$\boldsymbol{y}_i = g(\boldsymbol{x}_i, \boldsymbol{u}_i, \Theta_y) \tag{2.5b}$$

By unrolling the recurrence equation (2.5a), the loss function $L^k(\Theta)$ can be expressed as an explicit function of Θ. Thus, the optimization problem (2.4) can be solved with general optimization algorithms.

Generally speaking, the optimization problem (2.4) cannot be solved analytically for neural networks, nor can it be expected to be convex. Therefore, a numerical, iterative optimization algorithm must be adopted. In this work, the stochastic gradient descent (SGD) method is used for the training process. SGD is also the most commonly used training algorithm for RNNs. In SGD, the set of training parameters Θ is randomly initialized, then for each training sample, i.e. each time series, the parameters are updated using the following iterative equation:

$$\Theta_{new} = \Theta_{old} - r\frac{\partial L}{\partial \Theta}\bigg|_{\Theta_{old}} \tag{2.6}$$

where the learning rate r is a user controlled parameter which adjusts the balance between the speed and convergence of SGD. The gradient $\frac{\partial L}{\partial \Theta}$ can be explicitly calculated using the back-propagation through time (BPTT) algorithm, which is described below.

The gradient $\frac{\partial L}{\partial \Theta}$ can be expressed as a sum of the gradients at all the time steps:

$$\frac{\partial L}{\partial \Theta_x} = \sum_i \frac{\partial L}{\partial \hat{\boldsymbol{y}}_i}\frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{x}_i}\frac{d\boldsymbol{x}_i}{d\Theta_x} \tag{2.7a}$$

$$\frac{\partial L}{\partial \Theta_y} = \sum_i \frac{\partial L}{\partial \hat{\boldsymbol{y}}_i}\frac{\partial \hat{\boldsymbol{y}}_i}{\partial \Theta_y} \tag{2.7b}$$

In (2.7), $\frac{\partial L}{\partial \Theta_y}$ can be directly calculated from (2.5b); $\frac{\partial L}{\partial \Theta_x}$ needs to be calculated with

$$\frac{\partial L}{\partial \Theta_x} = \sum_i \frac{\partial L}{\partial \hat{\boldsymbol{y}}_i}\frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{x}_i}\left(\frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{x}_{i-1}}\frac{d\boldsymbol{x}_{i-1}}{d\Theta_x} + \frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{u}_i}\frac{\partial \boldsymbol{u}_i}{\partial \Theta_x} + \frac{\partial \boldsymbol{x}_i}{\partial \Theta_x}\right) \tag{2.8}$$

Since $\boldsymbol{u}_i$ is the independent input, $\frac{\partial \boldsymbol{u}_i}{\partial \Theta} = O$. On the other hand, the term $\frac{d\boldsymbol{x}_{i-1}}{d\Theta_x}$ can be expanded as a product of partial derivatives by applying the chain rule repeatedly. Therefore,

$$\frac{\partial L}{\partial \Theta_x} = \sum_i \frac{\partial L}{\partial \hat{\boldsymbol{y}}_i} \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{x}_i} \sum_{j=1}^{i} \left(\prod_{k=j+1}^{i} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{x}_{k-1}} \right) \frac{\partial \boldsymbol{x}_j}{\partial \Theta_x} \tag{2.9}$$

In (2.9), all partial derivative terms can be explicitly calculated from (2.5).

It is worth noting that in practical training tasks, the value of the time step h sometimes causes numerical difficulty in the training process. Specifically, an examination of (2.5a) reveals that the gradient $\frac{\partial L}{\partial \Theta_x}$ depends linearly on h; thus, too small an h can cause the gradient to become zero, while too large an h will cause the gradient to explode. In those situations, it would help to introduce a normalization parameter to the RNN equation (2.1a). With the normalization parameter τ, the general RNN equation can be expressed as

$$\dot{\boldsymbol{x}}(t) = \tau^{-1} \cdot f(\boldsymbol{x}(t), \boldsymbol{u}(t), \Theta_x) \tag{2.10a}$$

$$\boldsymbol{y}(t) = g(\boldsymbol{x}(t), \boldsymbol{u}(t), \Theta_y) \tag{2.10b}$$

Accordingly, for training with discrete-time data, the recurrence equation (2.5a) becomes

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + h\tau^{-1} \cdot f(\boldsymbol{x}_{i-1}, \boldsymbol{u}_i, \Theta_x) \tag{2.11}$$

In this way, the gradient $\frac{\partial L}{\partial \Theta_x}$ can be normalized with an appropriately chosen τ. An especially significant and natural choice of the normalization factor is $\tau = h$, which will often be used in this work. With $\tau = h$, the RNN equation (2.1a) becomes

$$\dot{\boldsymbol{x}} = h^{-1} \cdot f(\boldsymbol{x}, \boldsymbol{u}, \Theta_x) \tag{2.12}$$

and for training with time series data, the recurrence equation (2.11) becomes

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + f(\boldsymbol{x}_{i-1}, \boldsymbol{u}_i, \Theta_x) \tag{2.13}$$

2.3.1 Practical Details of the Training Setup

In this section, practical training settings used in this work are summarized.

As a common machine learning practice, the training data are separated into two subsets: the training set and validation set. The parameter updates use only data in the training set, and the validation set is used to evaluate overfitting. In this work, 80% of the training data are used in training, and 20% in validation, with the data randomly separated.

The training process performs parameter update according to (2.6) using all samples in the training set in a non-repetitive manner. In each training epoch, all training set samples are used for parameter update exactly once. After each epoch, the training error and validation error for the epoch are calculated with the entire training set and validation set according to (2.3). The training set is shuffled for the next epoch. The training process will terminate if the validation error becomes stagnant, i.e. stops improving significantly for more than a preset number of epochs, or the total number of epochs exceeds a preset maximum. In this work, the default numbers for those preset parameters are 100 for stagnancy, and 2000 for maximum.

The learning rate r in (2.6) is adaptively controlled. Numerous methods for controlling the SGD learning rate have been developed, and the AdaDelta [38] algorithm is chosen for this work. In addition, the learning rate is further reduced when the training error is detected to be plateaued for a long period (by default 20 epochs), similar to the learning rate annealing method introduced in [39].

In this work, the training program is written in Python, and open-source packages Theano and Keras are used.

2.4 Verilog-A Implementation of RNN

In Sections 2.1 to 2.3, the process of creating an RNN model for an electrical circuit is introduced. However, to use the RNN model in a general-purpose circuit simulator, its model equations need to be implemented in a form interpretable by the simulator. In this work, we choose to implement the RNN model using the Verilog-A language, a *de facto* industry standard for compact modeling.

2.4.1 Direct Verilog-A Implementation of Continuous-time RNN Equations

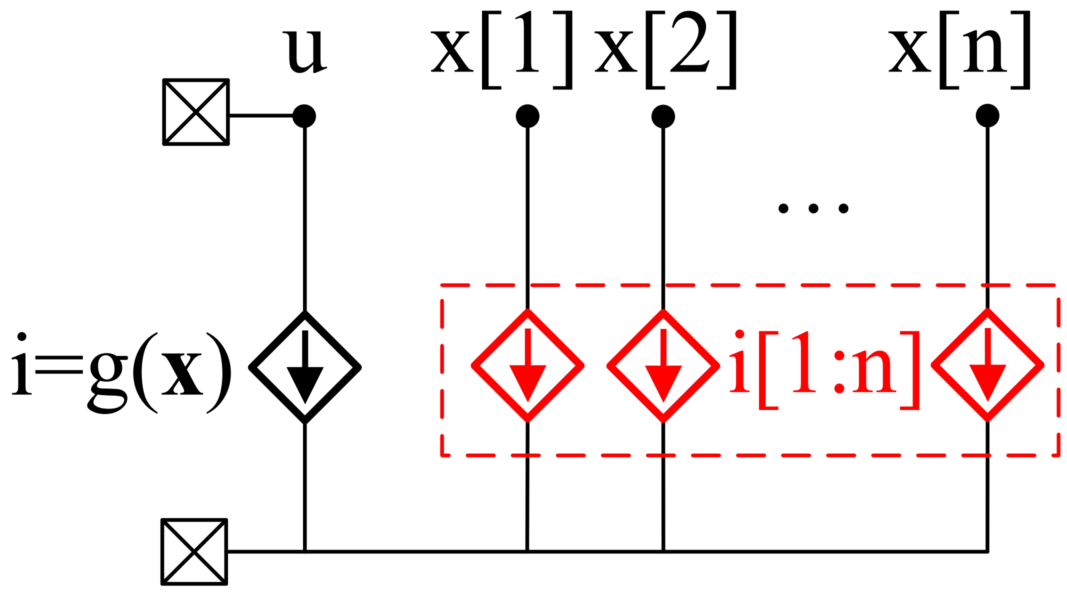


Figure 2.4: Schematic representation of the Verilog-A RNN model for a 1-port circuit.

In this section, a method of implementing (2.1) in Verilog-A is given. Figure 2.4 shows the Verilog-A netlist of a 1-port RNN model with the port being VICO. The RNN internal states $\boldsymbol{x} = x[1\text{:n}]$ are represented by internal nodes in Verilog-A. The state transition equation (2.1a) is implemented by the current branches labeled $i[1\text{:n}]$, with

$$i[1\text{:n}] = \boldsymbol{i} = \dot{\boldsymbol{x}} - f(\boldsymbol{x}, \boldsymbol{u})$$

where the quantity $\dot{\boldsymbol{x}}$ is implemented using the Verilog-A time-differential operator `ddt`. The output equation (2.1b) is implemented by the current branch labeled $i = g(\boldsymbol{x})$. During circuit simulation, KCL equation is formulated for all nodes. For each of the nodes x[1:n], only one current branch is connected, thus the KCL equations become i[1:n] $= \mathbf{0}$, which is equivalent to the RNN equation (2.1a).

It is simple to modify the netlist shown in Figure 2.4 for RNN models with multiple ports and/or ports with current as input and voltage as output. In both cases the internal nodes $\boldsymbol{x}$ and current branches i[1:n] do not need to be modified. For multi-port RNN, additional input nodes u and output branches $i = g(\boldsymbol{x})$ need to be added; for a port with current as input, the dependent current source $i = g(\boldsymbol{x})$ needs to be replaced by a dependent voltage source $v = g(\boldsymbol{x})$.

2.4.2 Verilog-A Implementation of Discrete-time RNN Equations

Since (2.5) is used directly in the training process, it may be reasonable to consider (2.5) as the fundamental RNN equations, and create the Verilog-A model by converting it to differential equations. This Verilog-A implementation method will be introduced in the next paragraph. Interestingly, it is observed that switching the Verilog-A implementation method from one to another does not cause significant change in model accuracy or numerical performance. The error introduced by the Verilog-A implementation is usually negligible compared with the error of the RNN model itself. Nevertheless, the Verilog-A implementation method introduced in Section 2.4.1 is preferred for theoretical reasons which will be addressed in Section 3.1.2.

To convert the discrete-time RNN equations to differential form, the input, output and hidden states of the RNN need to be replaced by their continuous-time counterparts and their time-derivatives. Essentially, the RNN equations (2.5) describe the dynamic behavior of the system at an arbitrary time step t_i. A first-order expansion of the continuous-time inputs and hidden states is taken at $t = t_{i-1} + \alpha h$. Here, $0 < \alpha < 1$ is a parameter introduced for model conversion, and h is the time step of the training data. The discrete-time RNN inputs and hidden states may then be approximated as:

$$\boldsymbol{u}_i = \boldsymbol{u} + (1-\alpha)h\dot{\boldsymbol{u}} \tag{2.14a}$$

$$\boldsymbol{x}_{i-1} = \boldsymbol{x} - \alpha h\dot{\boldsymbol{x}} \tag{2.14b}$$

$$\boldsymbol{x}_i = \boldsymbol{x} + (1-\alpha)h\dot{\boldsymbol{x}} \tag{2.14c}$$

Substituting (2.14) into the recurrence relation (2.5) yields

$$\boldsymbol{x} + (1-\alpha)h\dot{\boldsymbol{x}} = \boldsymbol{x} - \alpha h\dot{\boldsymbol{x}} + h \cdot f(\boldsymbol{x} - \alpha h\dot{\boldsymbol{x}}, \boldsymbol{u} + (1-\alpha)h\dot{\boldsymbol{u}}, \Theta_x) \tag{2.15a}$$

$$\boldsymbol{y} = g(\boldsymbol{x}, \boldsymbol{u}, \Theta_y) \tag{2.15b}$$

Next, (2.15) is implemented in Verilog-A in a way similar to that introduced in Section 2.4.1. The dependent current sources representing the RNN state transition equation in Figure 2.4 are now

$$\boldsymbol{i} = -h\dot{\boldsymbol{x}} + h \cdot f(\boldsymbol{x} - \alpha h\dot{\boldsymbol{x}}, \boldsymbol{u} + (1-\alpha)h\dot{\boldsymbol{u}}, \Theta_x) \tag{2.16}$$

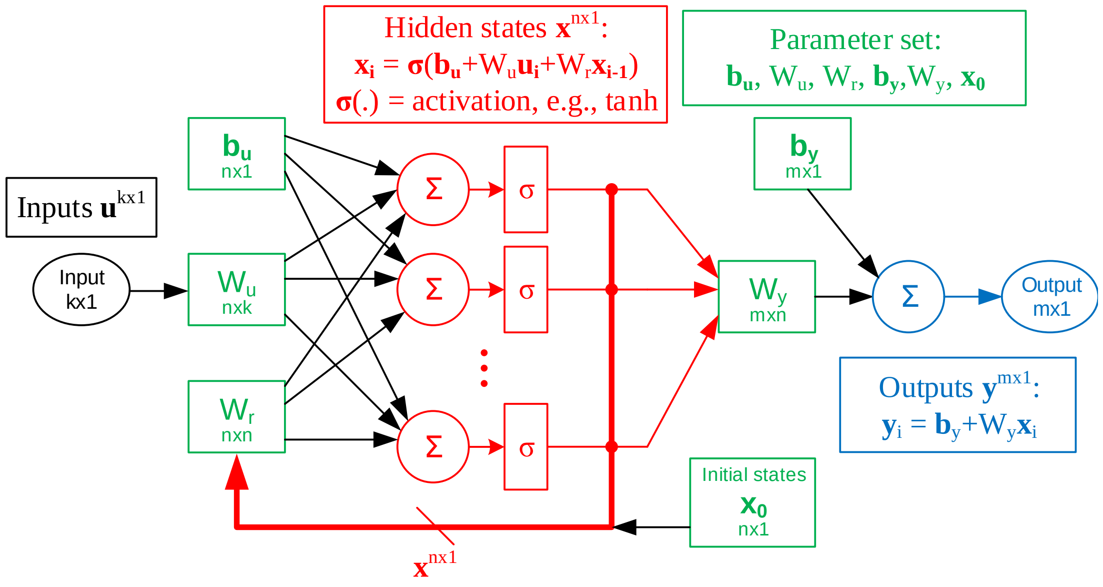
It has been found that the choice of the model parameter α affects the numerical stability of the Verilog-A model. This observation will be revisited in Section 3.1.2.

CHAPTER 3

RNN MODEL STRUCTURES

In this chapter, RNN model structures investigated in this work are introduced.

3.1 The Ordinary RNN



Hidden states $\mathbf{x}^{nx1}$:
$\mathbf{x_i} = \boldsymbol{\sigma}(\mathbf{b_u}+W_u\mathbf{u_i}+W_r\mathbf{x_{i-1}})$
$\boldsymbol{\sigma}(.)$ = activation, e.g., tanh
Parameter set:
$\mathbf{b_u}$, W_u, W_r, $\mathbf{b_y}$, W_y, $\mathbf{x_0}$
Inputs $\mathbf{u}^{kx1}$
Input kx1
$\mathbf{b_u}$ nx1
W_u nxk
W_r nxn
Σ
σ
$\mathbf{b_y}$ mx1
W_y mxn
Output mx1
Outputs $\mathbf{y}^{mx1}$:
$\mathbf{y}_i = \mathbf{b}_y+W_y\mathbf{x}_i$
Initial states $\mathbf{x_0}$ nx1
$\mathbf{x}^{nx1}$

Figure 3.1: Structure and model equations of a one-hidden-layer ordinary RNN with k inputs, m outputs, and n hidden states.

In this work, the term “ordinary RNN” is used to denote the most basic RNN model, characterized by its single hidden layer and linear output layer, as is shown in Figure 3.1. If the normalization constant $\tau = h$ is used where h is the time step of the training time series, the ordinary RNN has the following model

Table 3.1: Selected activation functions used in neural networks.

Name	Notation in this work	Explicit form
sigmoid	$\sigma_s(x)$	$\dfrac{1}{1+e^{-x}}$
hyperbolic tangent	$\tanh(x)$	$\dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
softplus	$\sigma_r(x)$	$\ln(1+e^x)$

equations:

$$\dot{\boldsymbol{x}} = h^{-1} \cdot [-\boldsymbol{x} + \tanh(W_r \boldsymbol{x} + W_u \boldsymbol{u} + \boldsymbol{b}_u)] \tag{3.1a}$$

$$\boldsymbol{y} = W_y \boldsymbol{x} + \boldsymbol{b}_y \tag{3.1b}$$

For the ordinary RNN, the model parameters are $W_u \in \mathbb{R}^{n_x \times n_u}$, $W_r \in \mathbb{R}^{n_x \times n_x}$, $\boldsymbol{b}_u \in \mathbb{R}^{n_x}$, $W_y \in \mathbb{R}^{n_y \times n_x}$, and $\boldsymbol{b}_u \in \mathbb{R}^{n_y}$.

In (3.1a), the activation function that introduces nonlinearity to the model is chosen to be hyperbolic tangent, but other functions can also be used in principle. The hyperbolic tangent function was chosen due to the intuitive understanding that the internal state variables of an electrical circuit, e.g. charges and fluxes, are mostly bipolarity quantities. Some common choices of activation function are listed in Table 3.1. Note that all the activation functions are strictly monotonic increasing, and their derivatives are all bounded between 0 and 1. In (3.1a), the activation function is applied element-wise to its argument, which is a vector.

For training with time series data, (2.13) is used to approximate the differential equation (3.1a) to a recurrence equation. The approximated equation is:

$$\boldsymbol{x}_i = \tanh\left(W_r \boldsymbol{x}_{i-1} + W_u \boldsymbol{u}_i + \boldsymbol{b}_u\right) \tag{3.2}$$

3.1.1 Zero-in-zero-out RNN

To adapt to the task of circuit modeling, a modification is introduced to the ordinary RNN equations (3.1) to accommodate for a common property of many circuits: For a physical sourceless circuit, if the condition $V = 0$ holds for all $t < 0$ at every terminal, i.e. the circuit is initialized to a rest state, then for a zero-input stimuli $V = 0$ for $t \geq 0$, the output must be $I = 0$ at every terminal due to conservation of energy. This property is referred to as zero-in-zero-out (ZIZO) in this work. The modified RNN equations below are introduced to account for the ZIZO property of circuit being modeled:

$$\dot{\boldsymbol{x}} = h^{-1}\left[-\boldsymbol{x} + \tanh\left(W_r\boldsymbol{x} + W_u\boldsymbol{u} + \boldsymbol{b}_u\right) - \tanh\boldsymbol{b}_u\right] \tag{3.3a}$$

$$\boldsymbol{y} = W_y\boldsymbol{x} \tag{3.3b}$$

Obviously, for the initial condition $\boldsymbol{x}(0) = \boldsymbol{0}$ and input waveform $\boldsymbol{u}(t) = \boldsymbol{0}$, the solution of differential equation (3.3) would be $\boldsymbol{x}(t) = \boldsymbol{0}$ and $\boldsymbol{y}(t) = \boldsymbol{0}$. This modified model has the same training complexity as the ordinary RNN (3.1).

For actual training, the ZIZO ordinary RNN equation (3.3a) is approximated with the following recurrence equation:

$$\boldsymbol{x}_i = \tanh\left(W_r\boldsymbol{x}_{i-1} + W_u\boldsymbol{u}_i + \boldsymbol{b}_u\right) - \tanh\boldsymbol{b}_u \tag{3.4}$$

It is easy to verify that this recurrence equation also satisfies ZIZO: When the hidden states vector $\boldsymbol{x}$ is initialized to $\boldsymbol{x}_0 = \boldsymbol{0}$ and the input is $\boldsymbol{u}_i = \boldsymbol{0}$ for all i, the solution to the recurrence equation (3.4) can be found to be $\boldsymbol{x}_i = \boldsymbol{0}$ for all i.

Below, (3.5) present an alternative (and simpler) set of discrete-time RNN equations that guarantee ZIZO:

$$\boldsymbol{x}_i = \tanh\left(W_r \boldsymbol{x}_{i-1} + W_u \boldsymbol{u}_i\right) \tag{3.5a}$$

$$\boldsymbol{y}_i = W_y \boldsymbol{x}_i \tag{3.5b}$$

This alternative equation provides ZIZO with initialization $\boldsymbol{x}_0 = \boldsymbol{0}$ since $\tanh(0) = 0$. However, the model described by (3.5) has an inconvenient symmetry that makes it unsuitable for many circuits. Suppose that for a specific input time series $\boldsymbol{u}_i = \boldsymbol{f}_u(i)$, the model predicts a hidden states time series $\boldsymbol{x}_i = \boldsymbol{f}_x(i)$ and $\boldsymbol{y}_i = \boldsymbol{f}_y(i)$. For the negative input time series $\boldsymbol{u}_i = -\boldsymbol{f}_u(i)$, it can be derived from mathematical induction that $\boldsymbol{x}_i = -\boldsymbol{f}_x(i)$ and $\boldsymbol{y}_i = -\boldsymbol{f}_y(i)$. A significant implication of this result is that any circuit modeled with (3.5) must have an I-V characteristic that is an odd function. Many circuits do not have this property, e.g. a single diode. Therefore, the model (3.5) cannot be used as a general circuit model.

3.1.2 Numerical Stability of Discrete-time Verilog-A Implementation

It is observed that for the ordinary and ZIZO ordinary RNN, if the Verilog-A model creation method introduced in Section 2.4.2 is used, numerical instability may occur depending the choice of the parameter α used for the recurrence-to-differential equation conversion (2.15). Specifically, non-convergence of transient simulation is observed when the value of α is close to 1. This phenomenon will be discussed using the ZIZO ordinary RNN as an example.

Consider the circuit netlist shown in Figure 3.2 with which a transient simulation is run. In Figure 3.2, the Verilog-A model netlist for the ZIZO ordinary RNN is represented by the elements inside the dashed box, and dependent current

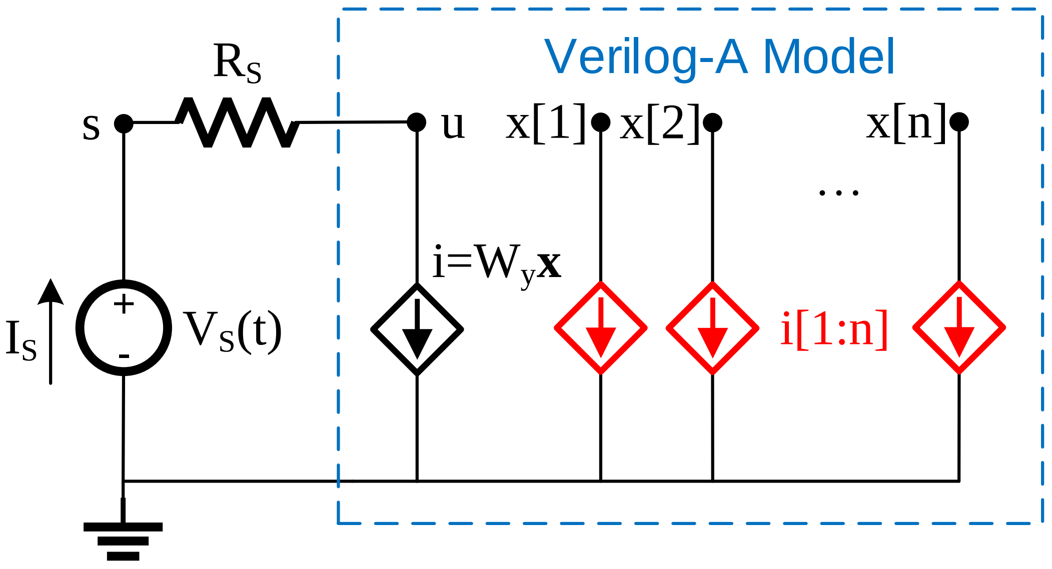


Figure 3.2: Circuit diagram for numerical stability analysis.

sources i[1:n] are defined by (2.16). Substituting the equation for ZIZO ordinary RNN (3.4), one obtains

$$\text{i[1:n]} = \boldsymbol{x} - (1-\alpha)h\dot{\boldsymbol{x}} - \tanh[W_r(\boldsymbol{x} - \alpha h\dot{\boldsymbol{x}}) + W_u(u - (1-\alpha)h\dot{u}) + \boldsymbol{b}_u] + \tanh \boldsymbol{b}_u \quad (3.6)$$

In the Verilog-A model, the variables $\boldsymbol{x}$ and u are represented by node voltages. In addition, to implement the time differential of a signal, e.g. $\dot{\boldsymbol{x}}$, virtual nodes are created in the Verilog-A model; those nodes are denoted as $\boldsymbol{d} = h\dot{\boldsymbol{x}}$ and $c = h\dot{u}$. Defining i[1:n] $= \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{d}, u, c)$ and assuming that the backward Euler method is applied in the transient simulation, the modified nodal analysis (MNA) equations representing the circuit netlist are as follows:

$$s = v_s \tag{3.7a}$$
$$i_s + \frac{s-u}{R_s} = 0 \tag{3.7b}$$
$$W_y\boldsymbol{x} + \frac{u-s}{R_s} = 0 \tag{3.7c}$$
$$\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{d}, u, c) = \boldsymbol{0} \tag{3.7d}$$
$$\frac{h\boldsymbol{x}}{\tau} - \boldsymbol{d} = \frac{h\boldsymbol{x}_{prev}}{\tau} \tag{3.7e}$$
$$\frac{hu}{\tau} - c = \frac{hu_{prev}}{\tau} \tag{3.7f}$$

In (3.7), s is the node voltage labeled in Figure 3.2, $\boldsymbol{x}_{prev}$ and u_{prev} are the solutions of $\boldsymbol{x}$ and u at the previous time step, and τ is the variable time step used in the transient simulation. Since (3.7d) is nonlinear, the circuit simulator will attempt to solve the set of equations iteratively using the Newton-Raphson method. Defining vector $\boldsymbol{X} = \{\boldsymbol{x}, \boldsymbol{d}, u, c\}$, the Newton iteration formula for (3.7d) may be written as

$$J_F^*\boldsymbol{X} = -\boldsymbol{F}(\boldsymbol{X}^*) + J_F^*\boldsymbol{X}^* \tag{3.8}$$

In (3.8), $\boldsymbol{X}^*$ is the value of $\boldsymbol{X}$ in the previous Newton-Raphson iteration, and J_F^* is the Jacobian of $\boldsymbol{F}(\boldsymbol{X})$ when $\boldsymbol{X} = \boldsymbol{X}^*$. The entire set of MNA equations can be written in the following matrix form, which is the same as the matrix generated by the stamp method in the circuit simulator:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & O & O \\ -1 & 1/R_s & -1/R_s & 0 & O & O \\ 0 & -1/R_s & 1/R_s & 0 & W_y & O \\ 0 & 0 & h/\tau & -1 & O & O \\ O & O & J_u^* & J_c^* & J_x^* & J_d^* \\ O & O & O & O & hI/\tau & -I \end{bmatrix} \begin{bmatrix} i_s \\ s \\ u \\ c \\ \boldsymbol{x} \\ \boldsymbol{d} \end{bmatrix} = \begin{bmatrix} v_s \\ 0 \\ 0 \\ hu_{prev}/\tau \\ -\boldsymbol{F}(\boldsymbol{X^*}) + J_F^*\boldsymbol{X^*} \\ h\boldsymbol{x}_{prev}/\tau \end{bmatrix} \tag{3.9}$$

Using (3.6), the Jacobians are found to be

$$J_u = \partial \boldsymbol{F}/\partial u = -M_O W_u \tag{3.10a}$$

$$J_c = \partial \boldsymbol{F}/\partial c = -(1-\alpha) M_O W_u \tag{3.10b}$$

$$J_x = \partial \boldsymbol{F}/\partial \boldsymbol{x} = I - M_O W_r \tag{3.10c}$$

$$J_d = \partial \boldsymbol{F}/\partial \boldsymbol{d} = (1-\alpha) I - \alpha M_O W_r \tag{3.10d}$$

where I is the identity matrix and M_O is the diagonal matrix specified in (3.11).

$$M_O = diag\left\{\tanh'[W_r(\boldsymbol{x^*} - \alpha \boldsymbol{d^*}) + W_u(u^* + (1-\alpha)c^*)]\right\} \tag{3.11}$$

In (3.11), $\tanh'$ denotes the derivative of the hyperbolic tangent function, i.e.

$$\tanh'(x) = \frac{d\tanh(x)}{dx} = \frac{1}{1+\tanh^2(x)}$$

If the model parameters W_u, W_r, and W_y are known, the only unknowns in the MNA matrix are the time step τ and the diagonal matrix M_O. The entries of matrix M_O depend on the value of intermediate variables during Newton-Raphson iterations and are impossible to know a priori. However, by treating the elements of M_O as random variables and using Monte Carlo simulation, the numerical stability of the transient simulation for a specific time step τ can be evaluated. In fact, a large set of randomly generated M_O represents a wide range of MNA matrices being solved in actual circuit simulations. Since the elements of M_O are derivatives of the hyperbolic tangent function, they are bounded by $[0, 1]$, and a uniform distribution is assumed for the Monte Carlo simulation.

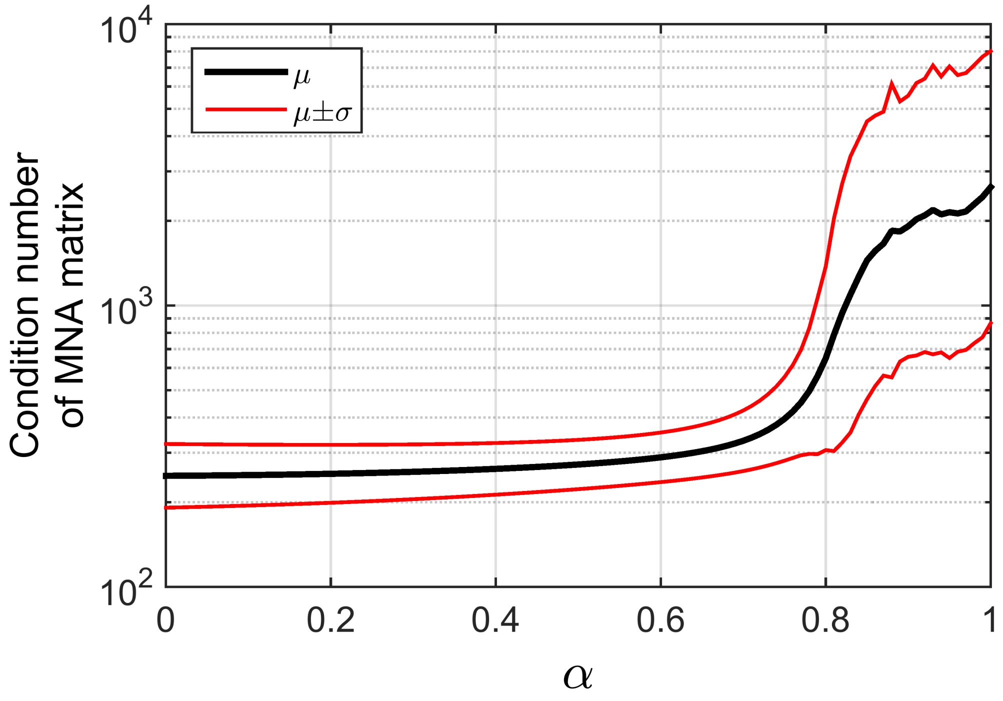


Figure 3.3: MNA matrix condition number from Monte Carlo simulation with random M_O.

The Monte Carlo simulation is done for an RNN and the relationship between the condition number of the MNA matrix and α is plotted in Figure 3.3. The transient simulation time step τ is taken to be 1/10 of the RNN time step h. The results of this exercise suggest that the condition number of the MNA matrix increases dramatically when α gets close to one. Moreover, the variance of the MNA matrix condition number also increases when α gets close to one, suggesting an increased likelihood of causing numerical instability.

Due to the numerical instability issue discussed in this section, it is in general recommended to use $\alpha = 0$ when the method introduced in Section 2.4.2 is used for the Verilog-A implemention of an RNN model. Accordingly, the differential equation implemented in the Verilog-A model becomes

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u} + h\dot{\boldsymbol{u}}, \Theta_x) \tag{3.12a}$$

$$\boldsymbol{y} = g(\boldsymbol{x}, \boldsymbol{u}, \Theta_y) \tag{3.12b}$$

This equation differs from the differential form RNN equations (2.1) only by the $h\dot{\boldsymbol{u}}$ term. It is observed that this term has negligible effect on the accuracy of the Verilog-A model. On the other hand, the inclusion of the $h\dot{\boldsymbol{u}}$ term causes violation of the causality principle. Therefore, from a theoretical point of view, it is recommended to directly use (2.1) for the Verilog-A model.

3.2 The Asymmetric Hopfield Network

Another RNN structure widely investigated in the past is the Hopfield network. The Hopfield network is a single-layer RNN like the ordinary RNN, but it adds a decay constant for each of the hidden states. If the normalization constant $\tau = h$ is used, the Hopfield network has the following model equations:

$$\dot{\boldsymbol{x}} = h^{-1}\left[-\Lambda \boldsymbol{x} + W_r \tanh\left(\boldsymbol{x} + W_u \boldsymbol{u} + \boldsymbol{b}_u\right)\right] \tag{3.13a}$$

$$\boldsymbol{y} = W_y \boldsymbol{x} + \boldsymbol{b}_y \tag{3.13b}$$

where $\Lambda \in \mathbb{R}^{n_x \times n_x}$ is a diagonal positive definite matrix representing the decay constants. In practice, the positiveness of those parameters can be achieved through applying a training constraint, or defining the parameters as an exponential of real numbers. Although in the original work about the Hopfield network

[40] the matrix W_r is assumed to be symmetric, in this work no such constraint is imposed.

Similar to the ordinary RNN, for training with time series data, (3.13a) is approximated with a recurrence equation according to (2.13):

$$\boldsymbol{x}_i = (I - \Lambda)\boldsymbol{x}_{i-1} + W_r \tanh\left(\boldsymbol{x}_{i-1} + W_u \boldsymbol{u}_i + \boldsymbol{b}_u\right) \tag{3.14}$$

In addition, when the ZIZO property applies to the circuit being modeled, a ZIZO version of the Hopfield network can be used:

$$\dot{\boldsymbol{x}} = h^{-1}\left[-\Lambda \boldsymbol{x} + W_r\left(\tanh(\boldsymbol{x} + W_u \boldsymbol{u} + \boldsymbol{b}_u) - \tanh(\boldsymbol{b}_u)\right)\right] \tag{3.15a}$$

$$\boldsymbol{y} = W_y \boldsymbol{x} \tag{3.15b}$$

and the recurrence equation for training with time series data becomes

$$\boldsymbol{x}_i = (I - \Lambda)\boldsymbol{x}_{i-1} + W_r\left[\tanh\left(\boldsymbol{x}_{i-1} + W_u \boldsymbol{u}_i + \boldsymbol{b}_u\right) - \tanh(\boldsymbol{b}_u)\right] \tag{3.16}$$

3.3 The Gated Recurrent Unit

Another RNN structure investigated in this work is the gated recurrent unit (GRU) [41]. The GRU is proposed as a simplified version of the long short-term memory (LSTM) network, and both the GRU and LSTM were developed to mitigate the vanishing gradient problem, which will be discussed in detail in Chapter 4. If the normalization constant $\tau = h$ is used, the GRU has the following model equations:

$$\boldsymbol{z} = \sigma_s\left(\boldsymbol{b}_z + U_z\boldsymbol{u} + R_z\boldsymbol{x}\right) \tag{3.17a}$$

$$\boldsymbol{r} = \sigma_s\left(\boldsymbol{b}_r + U_r\boldsymbol{u} + R_r\boldsymbol{x}\right) \tag{3.17b}$$

$$\boldsymbol{x} = h^{-1}(1 - \boldsymbol{z}) * \left[-\boldsymbol{x} + \tanh\left(\boldsymbol{b}_h + U_h\boldsymbol{u} + R_h(\boldsymbol{r} * \boldsymbol{x})\right)\right] \tag{3.17c}$$

$$\boldsymbol{y} = W_y\boldsymbol{x} + \boldsymbol{b}_y \tag{3.17d}$$

In (3.17), $\boldsymbol{z} \in \mathbb{R}^{n_x}$ and $\boldsymbol{r} \in \mathbb{R}^{n_x}$ are intermediate variables, σ_s is the sigmoid function listed in Table 3.1, and all subscript terms R_{-}, U_{-}, $\boldsymbol{b}_{-}$ and W_{-} are trainable model parameters. The operator $*$ denotes element-wise (Hadamard) multiplication of two vectors.

For training with time series data, (2.13) can be applied to (3.17c) to generate an approximate recurrence equation. Converting the intermediate variables $\boldsymbol{z}$ and $\boldsymbol{r}$ also to time series, the resultant recurrence equation can be expressed with three separate equations:

$$\boldsymbol{z}_i = \sigma_s\left(\boldsymbol{b}_z + U_z\boldsymbol{u}_i + R_z\boldsymbol{x}_{i-1}\right) \tag{3.18a}$$

$$\boldsymbol{r}_i = \sigma_s\left(\boldsymbol{b}_r + U_r\boldsymbol{u}_i + R_r\boldsymbol{x}_{i-1}\right) \tag{3.18b}$$

$$\boldsymbol{x}_i = \boldsymbol{z}_i * \boldsymbol{x}_{i-1} + (\boldsymbol{1} - \boldsymbol{z}_i) * \tanh\left[\boldsymbol{b}_h + U_h\boldsymbol{u}_i + R_h(\boldsymbol{r}_i * \boldsymbol{x}_{i-1})\right] \tag{3.18c}$$

Again, the GRU equations can also be modified to accommodate for the ZIZO property of electrical circuits. The ZIZO GRU equations are:

$$\boldsymbol{z} = \sigma_s \left(\boldsymbol{b}_z + U_z\boldsymbol{u} + R_z\boldsymbol{x}\right) \tag{3.19a}$$

$$\boldsymbol{r} = \sigma_s \left(\boldsymbol{b}_r + U_r\boldsymbol{u} + R_r\boldsymbol{x}\right) \tag{3.19b}$$

$$\boldsymbol{x} = h^{-1}(1-\boldsymbol{z}) * \left[-\boldsymbol{x} + \tanh\left(\boldsymbol{b}_h + U_h\boldsymbol{u} + R_h(\boldsymbol{r} * \boldsymbol{x})\right) - \tanh(\boldsymbol{b}_h)\right] \tag{3.19c}$$

$$\boldsymbol{y} = W_y\boldsymbol{x} \tag{3.19d}$$

3.4 A Modeling Example

An example RNN modeling task is presented in this section with definition of major performance figures-of-merit during the training process.

In this example, the circuit being modeled is chosen to be a nonlinear resistor and an inductor connected in series, which is also used in a test case in Section 4.2. Since the circuit is fairly simple, the model structure is chosen to be a ZIZO ordinary RNN with $n_x = 20$. The training data are generated using the random PWL method and converted to time series via interpolation.

During the training process, two performance indicators — the training error and the validation error — are closely monitored. The relative training and validation errors are defined by

$$E_r = \frac{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\frac{1}{L}\sum_{i=1}^{L}\left\|\boldsymbol{y}_i^k - \hat{\boldsymbol{y}}_i(\boldsymbol{u}^k,\Theta)\right\|^2}}{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\frac{1}{L}\sum_{i=1}^{L}\left\|\boldsymbol{y}_i^k\right\|^2}} \tag{3.20}$$

where E_r is the relative error, N is the total number of training/validation samples, L is the number of time points in each sample, $\boldsymbol{u}^k$ and $\boldsymbol{y}^k$ are the input and output time series of the k^{th} training/validation sample, and $\hat{\boldsymbol{y}}(\boldsymbol{u},\Theta)$ denotes the RNN prediction for input $\boldsymbol{u}$ with parameters Θ. The training and validation errors are calculated after each training epoch. In Figure 3.4, the evolution of the RNN training and validation error across the training epochs is plotted.

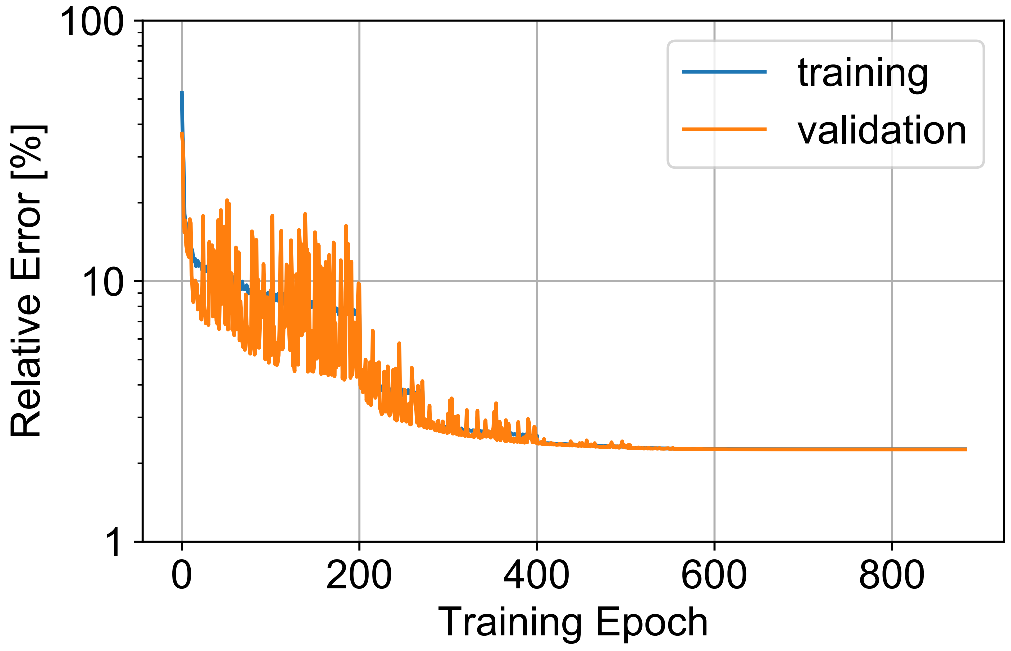


Figure 3.4: Evolution of training and validation errors across training epochs.

After the RNN model is trained, further evaluation is needed for the Verilog-A model implemented from the RNN. In model evaluation, transient simulation is run with the circuit being excited with random PWL voltage waveforms that are of much longer duration than those in the training dataset. By using a longer duration excitation, one can verify that the RNN model actually captures the dynamic behavior of the circuit rather than just replicating the known response of the circuit for a limited amount of time. This transient simulation will also be referred to as "evaluation simulation" in latter parts of this dissertation. Figure 3.5 compares the evaluation simulation result for the original circuit and RNN model. In a transient simulation, the voltage and current waveforms predicted by the RNN model can both deviate from waveforms obtained from a simulation with the original circuit, making it necessary to quantify the error associated with both waveforms. The following method is used to quantify the error between simulated

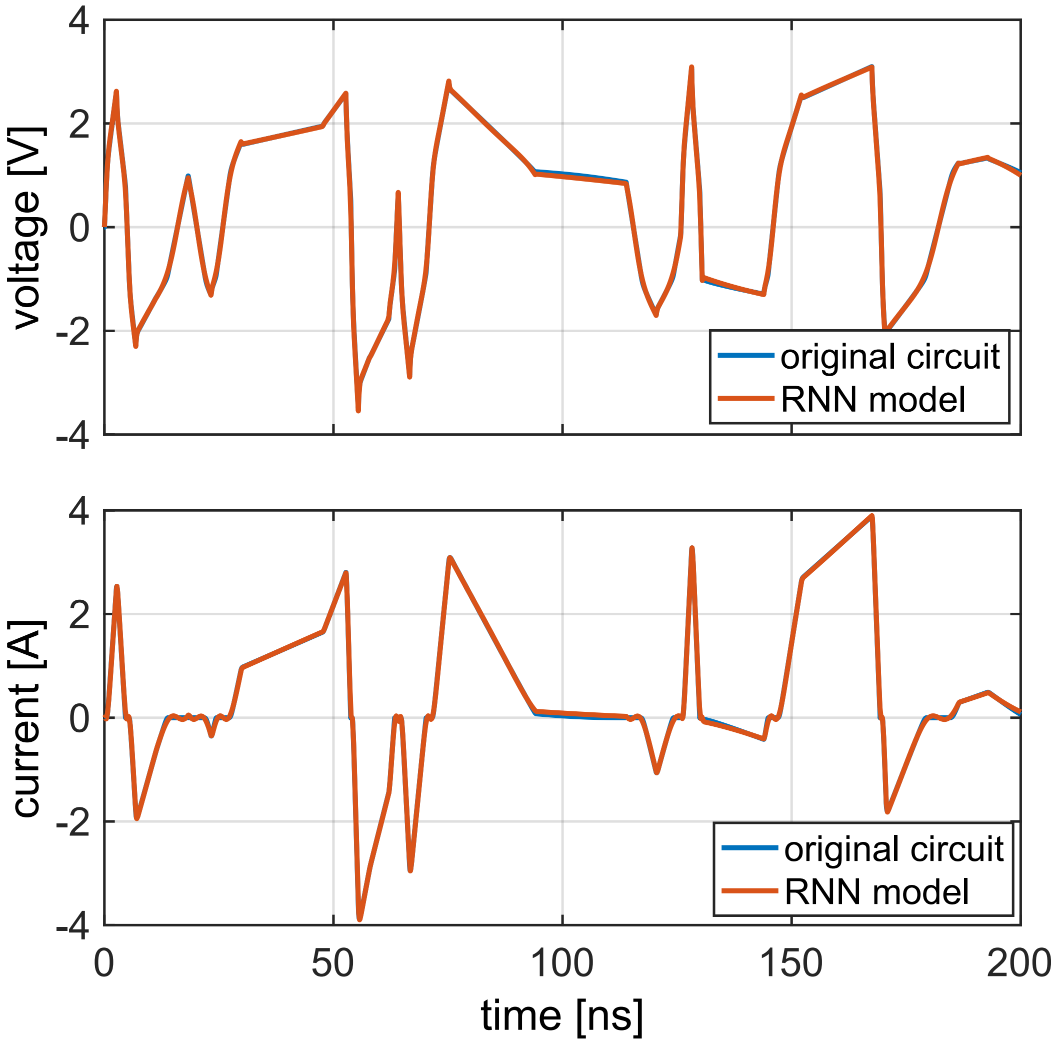


Figure 3.5: Transient simulation result for the original circuit and RNN model subjected to random PWL stimuli.

waveforms $\boldsymbol{y}$ from the original circuit and $\hat{\boldsymbol{y}}$ from the RNN model: First, both waveforms are linearly interpolated to the same time base of length L with a constant time step much smaller than that used for training data generation. This interpolation is necessary since the two waveforms are generated from separate transient simulations and thus may have values recorded at totally different sets of time instances. The interpolated waveforms $\boldsymbol{y}_i$ and $\hat{\boldsymbol{y}}_i$ are used in the following

Table 3.2: Model accuracy of RNN model of the modeling example.

Stimuli	Error
Training	Training 2.3% / Validation 2.3%
Evaluation	Voltage 1.14% / Current 1.17%

equation to calculate the relative error:

$$E_r = \frac{\sqrt{\frac{1}{L}\sum_{i=1}^{L}\|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|^2}}{\sqrt{\frac{1}{L}\sum_{i=1}^{L}\|\boldsymbol{y}_i\|^2}} \tag{3.21}$$

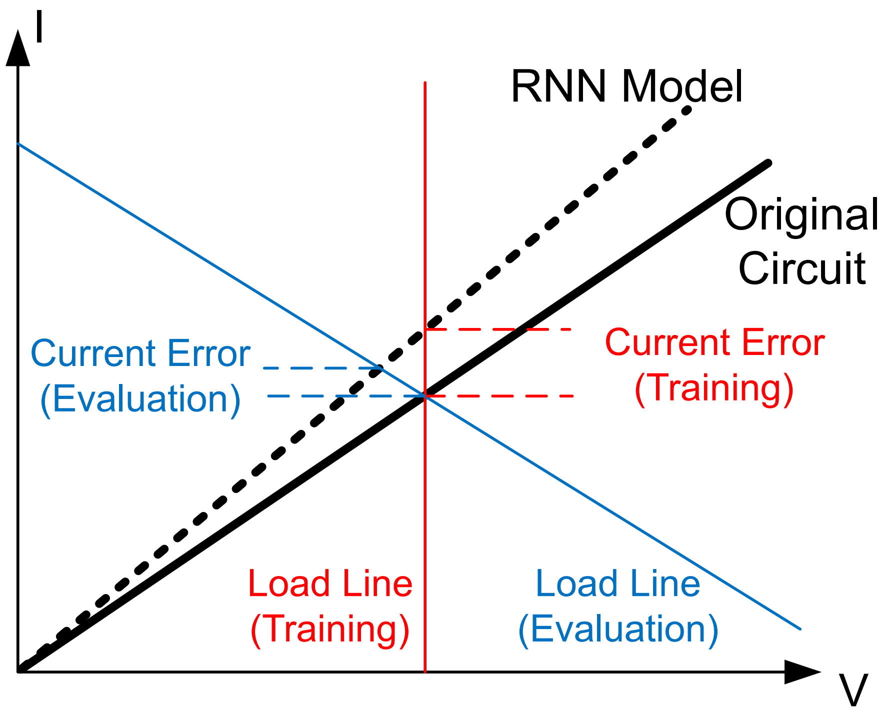


Figure 3.6: Illustration of difference in training error and evaluation error due to the load line effect.

For the modeling example, the relative errors in training, validation and evaluation are all listed in Table 3.2. As a rule of thumb used in this work, a model is considered accurate if the relative error is less than 5%, and acceptable if the relative error is between 5% and 10%. Using this rule, it can be concluded that the RNN model is accurate for this particular test circuit. Note that in Table 3.2, the evaluation error is smaller than the training/validation error. This is attributed to difference in the type of source used in the training and evaluation processes. In the training process, the RNN predicts the output current based on a fixed input voltage waveform. Equivalently, the model is connected to an ideal voltage source. In evaluation, however, the RNN model is sourced by a voltage source with a non-zero source resistance. As a result, the evaluation error is distributed between the input and output, and the error in each of the two waveforms can be smaller than the training error. A graphical illustration of this change in error due to load line is shown in Figure 3.6.

CHAPTER 4

THE VANISHING GRADIENT PROBLEM

4.1 Mathematical Description

One major drawback of the ordinary RNN comes from the issue known as the vanishing gradient problem. Equation (2.9) can be used to illustrate this issue. Rewrite (2.9) as

$$\frac{\partial L}{\partial \Theta_x} = \sum_i \sum_{j=1}^{i} \nabla_j^i(\Theta_x) \tag{4.1}$$

where $\nabla_j^i(\Theta_x)$ denotes the gradient of the output error at time step i with respect to the parameters Θ_x local to the time step j. Comparing with (2.9), it appears

$$\nabla_j^i(\Theta_x) = \frac{\partial L}{\partial \hat{\boldsymbol{y}}_i} \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{x}_i} \left(\prod_{k=j+1}^{i} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{x}_{k-1}} \right) \frac{\partial \boldsymbol{x}_j}{\partial \Theta_x} \tag{4.2}$$

The vanishing gradient problem arises when the parameter W_r in the ordinary RNN (3.1) is considered. From (4.2), it follows immediately that

$$\nabla_j^i(W_r) = \frac{\partial L}{\partial \hat{\boldsymbol{y}}_i} \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{x}_i} \left(\prod_{k=j+1}^{i} \frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{x}_{k-1}} \right) \frac{\partial \boldsymbol{x}_j}{\partial W_r} \tag{4.3}$$

The partial derivative in the product term of (4.3) can be calculated as:

$$\frac{\partial \boldsymbol{x}_k}{\partial \boldsymbol{x}_{k-1}} = diag(\tanh'(\boldsymbol{o}_k))W_r \tag{4.4}$$

where $\boldsymbol{o}_i = \boldsymbol{b}_u + W_r \boldsymbol{x}_{i-1} + W_u \boldsymbol{u}_i$, and $diag(\boldsymbol{x})$ denotes the diagonal matrix formed from vector $\boldsymbol{x}$. Substituting (4.4) into (4.2), one obtains

$$\nabla_j^i(W_r) = \frac{\partial L}{\partial \hat{\boldsymbol{y}}_i} \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{x}_i} \left(\prod_{k=j+1}^{i} diag(\tanh'(\boldsymbol{o}_k)) W_r \right) \frac{\partial \boldsymbol{x}_j}{\partial W_r} \tag{4.5}$$

For certain matrices W_r, the spectrum radius (i.e. maximum magnitude of eigenvalues) of the matrix inside the product operator in (4.5), $diag(\tanh'(\boldsymbol{o}_k))W_r$, can be uniformly less than 1 for all possible states $\boldsymbol{o}_k$. When this happens, the absolute value of the gradient $\nabla_j^i(W_r)$ will become zero when $i - j$ is sufficiently large. This zeroing of the long-term gradient is called the vanishing gradient problem. The vanishing gradient problem happens frequently since for all common activation functions used for neural networks, the derivative $\tanh(x)$ is bounded between 0 and 1. Especially, for the hyperbolic tangent function, the derivative $\tanh'(x)$ approaches zero when the value of $\tanh(x)$ approaches ± 1.

As a result, the gradient $\frac{\partial L}{\partial W_r}$ will only have contributions from short-term gradient terms, i.e. $\nabla_j^i(W_r)$ for which $i - j$ is not large. This can pose a training problem since some systems being modeled can have long-term input-output dependences — inputs having significant lingering effects many time steps later. Suppose for a specific system that an input at time step j causes an output response to occur at a later time step i. The RNN can only be trained to learn this dependence from the gradient update $\nabla_j^i(W_r)$. If this gradient update becomes zero due to the vanishing gradient problem, the long-term dependence will never be learned, and the information of this long-term dependence will be lost in the training process.

It may seem that an appropriate choice of the training data time step h could eliminate the vanishing gradient problem. Indeed, use of large h reduces the severity of the vanishing gradient problem since the same long-term dependence can be covered with fewer time steps, so that there are fewer terms in the product term in (4.5), making it less likely to reach zero. However, for systems that are expected to be excited by fast transients, using a large time step can significantly degrade the model accuracy. This is evident from the sampling theorem: If the training data time step is chosen to be h, the interpolated training data will contain no information about the circuit's behavior for stimuli with a frequency higher than $f_0 = \frac{1}{2h}$. Therefore, the trained model also does not contain any information about the model behavior when excited with stimuli with a frequency higher than f_0. In fact, considering the fact that most circuits being modeled in this work are nonlinear, they should generate spurious response that contains frequency content several times higher than the base frequency of a stimulus. Therefore, the time step h should be chosen to be several times higher than the expected maximum stimulus frequency.

4.2 Test Case: Nonlinear Resistor with Reactive/Delay Element

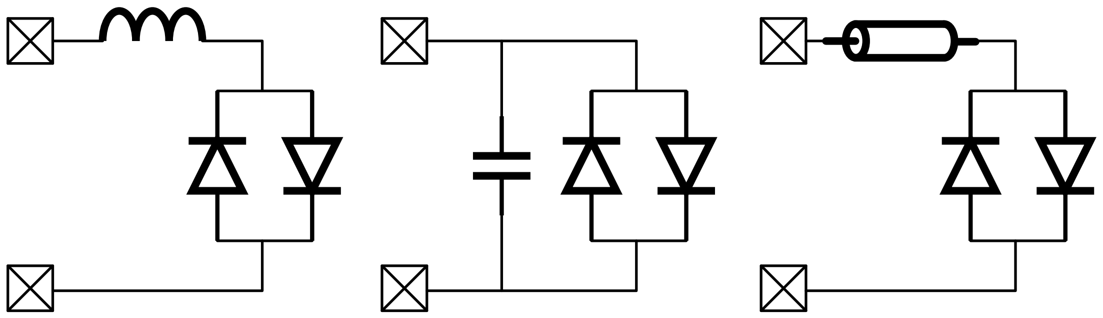


Figure 4.1: Test circuits used in the investigation of vanishing gradient problem.

Circuit modeling experiments are carried out to demonstrate the impact of the vanishing gradient problem. Three simple test circuits illustrated in Figure 4.1 are used for this experiment. All three test circuits contain the same nonlinear

Table 4.1: List of vanishing gradient test cases.

Settling time	Inductance	Capacitance	T-line delay
1 ns	0.18 nH	0.5 nF	0.25 ns
3 ns	0.54 nH	1.5 nF	0.75 ns
10 ns	1.78 nH	5 nF	2.5 ns
30 ns	5.36 nH	15 nF	7.5 ns
100 ns	17.8 nH	50 nF	25 ns

resistor, which is realized with two identical diodes connected in parallel facing different directions. The diode is designed to have no memory with the static I-V characteristics:

$$I_d = I_s \exp\left(\frac{q(V_d - R_s I_d)}{kT} - 1\right) \tag{4.6}$$

where V_d and I_d are the voltage across and current through the diode. Memory is introduced to the three test circuits by adding a single reactive or delay element to each circuit: an inductor in series, a capacitor in parallel, and an ideal transmission line (T-line) in series, respectively. Since the vanishing gradient problem becomes more severe for systems with longer memory, it is expected that the model accuracy will deteriorate when the inductance, capacitance, and T-line delay in the test circuits become larger, and the time step of the training data h is kept unchanged. To facilitate the comparison between the three test cases, the settling time is defined as a measure of the circuit memory. For each test circuit, simulation with the voltage across the test circuit being a step function is performed. From the simulation result, the settling time is defined as the time it takes for the current through the circuit to settle to less than 1% compared with its steady-state value.

Training data are generated using the random PWL method described in Section 2.2. The values of the components used in the actual netlists are listed in Table 4.1. All training samples extracted from circuit simulation are waveforms with a length of 100 ns, and the waveforms are interpolated with a fixed time step $h = 50$ ps to form the time series data directly used in training. In total, five sets of training data are generated for each test circuit.

For each set of training data, six different RNN models are trained. The three model structures — ZIZO ordinary RNN, ZIZO Hopfield network, and GRU, each with a hidden state vector dimension of 20 — are used to fit the training data. For each model structure, the circuit is configured as VICO or CIVO to train two different models. The results of all training experiments are groups with the type of memory element and input/output configuration, and are listed in Tables 4.2-4.4.

Table 4.2: Accuracy (in terms of Training Error/Validation Error of the output quantity) of RNN models trained for nonlinear resistor & series inductor.

Port I/O	Settling time	ZIZO Ordinary	ZIZO Hopfield	GRU
VICO	1 ns	2.3% / 2.3%	0.5% / 0.6%	0.3% / 0.3%
	3 ns	6.5% / 6.4%	3.8% / 4.1%	0.4% / 0.4%
	10 ns	9.4% / 9.4%	7.9% / 7.4%	0.8% / 0.8%
	30 ns	30.2% / 31.6%	14.1% / 12.7%	1.5% / 2.0%
	100 ns	16.4% / 11.2%	43.0% / 50.7%	6.2% / 9.2%
CIVO	1 ns	9.2% / 9.3%	9.4% / 9.2%	8.9% / 9.6%
	3 ns	8.9% / 8.7%	7.9% / 8.6%	7.7% / 7.5%
	10 ns	9.3% / 8.8%	6.1% / 6.7%	5.7% / 6.7%
	30 ns	12.6% / 12.5%	4.5% / 5.2%	6.2% / 5.8%
	100 ns	14.9% / 16.9%	11.3% / 15.1%	9.2% / 9.5%

Table 4.3: Accuracy (in terms of Training Error/Validation Error of the output quantity) of RNN models trained for nonlinear resistor & parallel capacitor.

Port I/O	Settling time	ZIZO Ordinary	ZIZO Hopfield	GRU
VICO	1 ns	3.3% / 2.8%	1.2% / 1.3%	0.5% / 0.5%
	3 ns	3.7% / 3.9%	Training diverges	0.9% / 0.9%
	10 ns	6.1% / 5.1%	1.0% / 1.0%	1.6% / 1.7%
	30 ns	11.5% / 13.8%	4.2% / 3.9%	2.2% / 2.5%
	100 ns	43.2% / 64.5%	13.6% / 22.8%	14.2% / 27.7%
CIVO	1 ns	2.6% / 2.7%	2.0% / 1.9%	0.4% / 0.4%
	3 ns	6.0% / 6.2%	5.5% / 5.0%	0.7% / 0.6%
	10 ns	5.7% / 5.8%	5.4% / 5.0%	1.8% / 1.8%
	30 ns	10.8% / 11.1%	8.0% / 9.4%	3.2% / 3.3%
	100 ns	76.6% / 74.5%	79.2% / 78.8%	14.4% / 16.8%

Table 4.4: Accuracy (in terms of Training Error/Validation Error of the output quantity) of RNN models trained for nonlinear resistor & series T-line.

Port I/O	Settling time	ZIZO Ordinary	ZIZO Hopfield	GRU
VICO	1 ns	2.8% / 2.8%	1.0% / 1.0%	0.8% / 1.0%
	3 ns	6.5% / 6.6%	2.7% / 2.8%	2.7% / 2.7%
	10 ns	29.3% / 28.3%	29.1% / 28.5%	28.9% / 29.0%
	30 ns	28.1% / 28.2%	28.0% / 27.8%	28.3% / 27.2%
	100 ns	22.0% / 22.5%	21.9% / 22.8%	22.3% / 21.4%
CIVO	1 ns	2.4% / 2.5%	1.3% / 1.3%	2.8% / 2.8%
	3 ns	6.1% / 5.9%	2.6% / 2.4%	2.2% / 2.5%
	10 ns	29.5% / 29.0%	29.5% / 29.5%	29.4% / 29.3%
	30 ns	28.4% / 28.5%	28.5% / 27.3%	28.4% / 28.4%
	100 ns	22.2% / 22.3%	22.4% / 21.5%	22.2% / 22.6%

From Tables 4.2-4.4, it is evident that the performances of all different RNN structures become worse with longer circuit settling time. This is consistent with the theory of vanishing gradient problem causing diminished accuracy. For a circuit with longer settling time, more information will be lost due to the vanishing gradient, causing the prediction error of the RNN to increase. On the other hand, comparing all the results between different model structures, it appears that the GRU is most robust for training of systems with long-term memory. The Hopfield network performs worse than GRU, and the ordinary RNN is the worst. This observation is expected since the GRU is specifically designed to alleviate the vanishing gradient problem. Unfortunately, even for the GRU, training error is still an increasing function of the circuit settling time, and the error can eventually become intolerable. It can also be observed from the results that different types of memory cause problems of different severity. For the systems with inductor and capacitor, the model accuracy gradually degrades with increasing settling time, and there are observable differences between the three structures. For the system with the transmission line, it appears that the training will completely fail for all structures once the delay is long enough. This result suggests that systems with pure delay are the most difficult to train for RNN structures, while systems with long decaying time constants can cause problems for the training algorithm, but using a GRU alleviates the problem. Note that systems with pure delay cannot be expressed with state-space equations with a finite number of states. Therefore, it is expected that RNN as a state-space system approximator has reduced performance for such systems.

It appears from the result that using VICO or CIVO does lead to difference in the model accuracy. This difference is especially prominent for the system with inductor, as the RNN models with voltage input outperform the current input ones by a large margin. For the systems with capacitor and transmission line, it is hard to conclude if one input configuration is necessarily better than the other.

This observation is hypothetically attributed to a combined effect of the reactive element and the nonlinear resistor. The dynamic behavior of the RL and RC systems can be easily described by a differential equation if the resistor in the system is assumed linear:

$$V = L\dot{I} + RI \tag{4.7a}$$

$$I = C\dot{V} + R^{-1}V \tag{4.7b}$$

where (4.7a) represents the series RL and (4.7b) represents the parallel RC. From those two equations, the sensitivity of voltage or current subject to a fast transient change in the other can be evaluated. For the RL system, I is not sensitive to a fast change in V because the $\dot{I}$ term acts like an integrator. On the other hand, V can be sensitive to a fast change in I especially if the resistance R is large. Similarly, for the RC system V is not sensitive to fast change in I, but I can be sensitive to fast change in V if R is small. Consider the behavior of the nonlinear resistor which consists of two diodes connected in anti-parallel. Its differential resistance will never be small, but can be large when the voltage is close to zero, when both diodes are in the off state. This indicates that for the nonlinear RL system, the voltage can be potentially too sensitive to the current, making the RNN with CIVO configuration difficult to train.

CHAPTER 5

STABILITY OF RNN MODELS

It is highly undesirable for a behavioral model to incorrectly predict that a stable system has an unstable response. However, as a class of black-box model, there is no guarantee that an RNN would be stable even if trained with data collected from a stable system. It is always possible to generate multiple models for the same system, test each and discard the unstable ones, but this method is highly inefficient. In this chapter, the RNN model structures are analyzed mathematically and certain conditions for stability are derived. In general, those conditions create a constraint on the RNN model parameters, such that if the training process is carried out with those constraints, the trained model can be guaranteed to be stable.

5.1 Definition of Model Stability

All RNN structures introduced in this work are in the form of a nonlinear state-space system. In general, the model can be written as

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{5.1a}$$

$$\boldsymbol{y}(t) = g(\boldsymbol{x}(t)) \tag{5.1b}$$

This work utilizes asymptotic stability; an asymptotically stable system will always converge to a constant equilibrium point for a constant input. Consequently, a circuit model in the form of (5.1) that is asymptotically stable cannot predict an oscillatory response to constant stimuli. Thus, for any circuit other than an oscillator, it is desired to enforce asymptotic stability when creating its model.

A formal definition of the asymptotic stability of the nonlinear system (5.1) is as follows [42]. Suppose that the constant input $\boldsymbol{u}(t) = \boldsymbol{U}$ and hidden state $\boldsymbol{X}$ satisfy $f(\boldsymbol{X}, \boldsymbol{U}) = 0$, i.e., $\boldsymbol{x} = \boldsymbol{X}$ is an equilibrium point of the system for input $\boldsymbol{u} = \boldsymbol{U}$. The equilibrium point is asymptotically stable if there exists $\epsilon > 0$ such that for all $|\boldsymbol{x}_0 - \boldsymbol{X}| < \epsilon$, the solution to differential equation (5.1a) with input $\boldsymbol{u} = \boldsymbol{U}$ and initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$ satisfies

$$\lim_{t\to\infty} \boldsymbol{x}(t) = \boldsymbol{X} \tag{5.2}$$

If condition (5.2) can be satisfied for all $\epsilon > 0$, the equilibrium point is said to be globally asymptotically stable. Furthermore, if all equilibrium points satisfying $f(\boldsymbol{X}, \boldsymbol{U}) = 0$ for all $\boldsymbol{U} \in \mathbb{R}^{n_u}$ are globally asymptotically stable, then system (5.1) is said to be absolutely stable (ABST). Clearly, the output of an ABST system will converge to a constant for any constant input.

For circuit simulation, the RNN model will be connected to models of other circuits in the full system being simulated. Potentially, more than one RNN model can be used in the same simulation. It is important to note that a system that consists of ABST systems connected in cascade is also ABST [43]. This means that a large circuit represented by cascaded ABST models cannot display nonphysical instability during circuit simulation. However, the ABST property has limitations in its application; most significantly, if an ABST system is connected in feedback, its stability is no longer guaranteed. Unfortunately, for RNN circuit models in which both the voltage and current at a single port are treated as variables — one as input and the other as output — the feedback connection is inherent. For example, consider an RNN model of a 1-port circuit in which the port voltage v is the input and current i is the output. Connecting a load resistor R to the port introduces feedback between the input and output, as given by the relation $v = Ri$. In fact, any circuitry connected to this port, other than an ideal voltage source, will introduce feedback to the RNN model. As a result, even if the RNN model is guaranteed to be ABST, no such statement can be made about the stability of the full system to be analyzed in a practical circuit simulation. Thus, if an RNN circuit model contains any IO-ports, a stability analysis must consider the source/load circuitry connected to those ports.

In theory, there are infinite varieties of source/load circuitry, and a stability analysis that covers all alternatives is infeasible. However, there are certain practical source/load configurations that can be analyzed. In particular, if the combined system of the source/load and the RNN can be expressed explicitly as a state-space system, the stability analysis can be carried out using the Lyapunov method. For example, consider a voltage source with series resistance and shunt capacitance. The output characteristic of that source is described by

$$i_{out} = \frac{V_s - v_{out}}{R} - C\frac{dv_{out}}{dt} \tag{5.3}$$

Further consider a multi-port RNN model whose input-output ports are of the type voltage-input current-output, and each of those ports is connected to a source that is described by (5.3). A comparison of (5.1) and (5.3) indicates that at the j$^{\text{th}}$ port, $v^j_{out} = u^j$ and $i^j_{out} = y^j$, and the combined system can be represented by:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \tag{5.4a}$$

$$\dot{\boldsymbol{u}} = (RC)^{-1}(\boldsymbol{V}_s - \boldsymbol{u}) - C^{-1}g(\boldsymbol{x}) \tag{5.4b}$$

In (5.4), R and C are diagonal matrices whose entries are the output resistances and capacitances of the sources connected to each port. The combined system described by (5.4) is autonomous with state variables $[\boldsymbol{x}\ \boldsymbol{u}]^T$, so one can define its asymptotic stability with (5.2). Furthermore, the combined system is ABST if it satisfies global asymptotic stability for all $\boldsymbol{V}_s$. Clearly, the stability of the system (5.4) depends on the source resistances and capacitances R and C. The designer should identify the range of R and C values that may be encountered in practice; furthermore, one can leverage expert knowledge about the characteristics of the circuit being modeled to identify the ranges of R and C for which the system should be stable. Then, if the combined system (5.4) is ABST for all R and C in that range, the RNN model can be identified as stable for this particular type of source. The preceding analysis is specific to one (often-encountered) model of a voltage source, and needs to be performed for different kinds of sources/loads on a case-by-case basis.

In this work, stability analysis is carried out for sources and loads that have a purely real output impedance and are connected port-wise to the RNN model. Sources connected to voltage-input ports are represented by Thevenin equivalent circuits and sources connected to current-input ports by Norton equivalent circuits. It follows that the feedback can be described using $\boldsymbol{u} = \boldsymbol{V} - R\boldsymbol{y}$, where $\boldsymbol{V}$ represents the Thevenin source voltage or Norton source current and R represents the Thevenin source resistance or Norton source conductance. The full, autonomous feedback system is then

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}(t), \boldsymbol{V} - Rg(\boldsymbol{x})) \tag{5.5}$$

An RNN model is said to be ABST for port-wise resistive feedback if the nonlinear system (5.5) is globally asymptotically stable for all $\boldsymbol{V}$ and diagonal positive definite R.

5.2 Conditions for Stability

In Section 5.1, stability is defined for RNN models as general nonlinear systems. In practice, the stability of different RNN structures needs to be analyzed case-by-case.

First of all, an observation can be made that the ordinary RNN can be considered as a special case of the Hopfield network. It may seem on examination that the ordinary RNN (3.1a) and Hopfield network (3.13a) are different since the ordinary RNN contains an activation of linearly reconnected states $\tanh(W_r\boldsymbol{x})$, while the Hopfield network contains a linearly reconnected activation of the states $W_r \tanh(\boldsymbol{x})$. However, for the ordinary RNN, one can make the transform of states $\bar{\boldsymbol{x}} = W_r\boldsymbol{x}$ so that (3.1a) becomes

$$\dot{\bar{\boldsymbol{x}}} = h^{-1}\left[-\bar{\boldsymbol{x}} + W_r \tanh(\bar{\boldsymbol{x}} + W_u\boldsymbol{u} + \boldsymbol{b}_u)\right] \tag{5.6}$$

After such a transform of states, the ordinary RNN becomes a Hopfield network with the state-wise decay constant $\Lambda = I$.

There is a body of prior research that addresses the stability conditions for a Hopfield network. However, for a Hopfield network with arbitrary parameter W_r, no sufficient and necessary condition for ABST has been established. In [44], it is shown that for Hopfield networks with a normal W_r matrix, i.e. $W_rW_r^T = W_r^TW_r$, a sufficient and necessary condition for ABST would be $max(Re\left[\lambda_e(W_r)\right]) < 0$, where $\lambda_e(A)$ represents the eigenvalues of matrix A. On the other hand, in [45, 46] it is demonstrated that if no constraint is imposed on W_r, a sufficient condition for the ABST of the Hopfield network (3.13) is $W_r - \Lambda \in \mathcal{L}$, where $\mathcal{L}$ is the set of Lyapunov diagonally stable (LDS) matrices, defined as

$$\mathcal{L} = \left\{ A \,|\, \exists D = diag(\boldsymbol{d}) > 0 : A^T D + DA < 0 \right\} \tag{5.7}$$

where $M > 0$ or $M < 0$ means that the symmetric matrix M is positive definite or negative definite, respectively. With either stability condition enforced, the resultant model space is only a subset of the set of all stable networks. In this work, we mainly focus on the LDS condition. With a stability condition identified, a corresponding constraint can be introduced into the training process to enforce stability. This is done by introducing a regularization term into the loss function that drives the model parameters toward the stable region.

It is not straightforward to construct a regularization term that corresponds to the LDS condition (5.7). However, if the stability criterion is further constrained to be $W_r + W_r^T - 2\Lambda < 0$, a regularization term that is a function of the eigenvalues of the symmetric matrix $W_r + W_r^T - 2\Lambda$ can be introduced. Setting $D = I$ in (5.7), one may easily verify that $W_r + W_r^T - 2\Lambda < 0$ is a sufficient condition for $W_r - \Lambda \in \mathcal{L}$. For an ordinary RNN, $\Lambda = I$, thus the stability condition becomes $W_r + W_r^T - 2I < 0$; this constraint can also be directly derived using the Lyapunov method, the details of which are given in Appendix A. The regularization term shown below is added to the loss function to penalize any positive eigenvalue of the matrix $W_r + W_r^T - 2\Lambda$:

$$R(\Theta) = \sum \sigma_r \left[\lambda_e (W_r + W_r^T - 2\Lambda) \right] \tag{5.8}$$

Above, $\lambda_e(M)$ denotes the eigenvalues of the symmetric matrix M, and σ_r is the softplus function listed in Table 3.1. With regularization, the optimization problem (2.4) becomes

$$\Theta_{best} = \arg\min_{\Theta} \sum_{k=1}^{N} \left[L^k(\Theta) + \gamma R(\Theta) \right] \tag{5.9}$$

where γ is a proportionality factor whose value is tuned to ensure model stability with only a minimal sacrifice in model accuracy.

As is discussed in Section 5.1, it is further desired to find conditions that guarantee the RNN model to be ABST for port-wise resistive feedback, as described in the preceding section. For an RNN, the feedback equation in (5.5) can be written as

$$\dot{\boldsymbol{x}} = h^{-1} \left[-\boldsymbol{x} + \tanh\left((W_r - W_u R W_y)\boldsymbol{x} + W_u \boldsymbol{V} - W_u R \boldsymbol{b}_y + \boldsymbol{b}_u \right) \right] \tag{5.10}$$

Equation (5.10) has the same functional form as (3.1a), and represents an ordinary RNN with different parameters. From the analysis of ABST for ordinary RNN, it immediately follows that the nonlinear system (5.10) is guaranteed to be globally asymptotically stable for all $\boldsymbol{V}$ if $W_r - W_u R W_y - I \in \mathcal{L}$. Therefore, a sufficient stability condition for an RNN with port-wise resistive feedback is

$$\forall R = diag(\boldsymbol{r}) > 0 : W_r - W_u R W_y - I \in \mathcal{L} \tag{5.11}$$

If there exists a diagonal positive definite matrix D such that $W_y^T = D W_u$, then a sufficient condition of (5.11) can be formulated in terms of matrix eigenvalues and introduced into training as a regularization. Specifically, condition (5.11) can be satisfied for all $R > 0$ if $D(W_r - I) + (W_r - I)^T D < 0$.

Proof. Let $A = W_r - I$. From $W_y^T = D W_u$, one obtains

$$D(A - W_u R W_y) + (A - W_u R W_y)^T D = DA + A^T D - 2 W_y R W_y^T$$

Since R is positive definite, $W_y R W_y^T$ is also positive definite. Therefore, if $DA + A^T D < 0$, it follows that $DA + A^T D - 2W_y R W_y^T < 0$ for all $R > 0$. Thus, (5.7) is satisfied for $A - W_u R W_y$. $\square$

Similar to (5.8), a regularization term penalizing positive eigenvalues of matrix $D(W_r - I) + (W_r - I)^T D$ can be introduced to enforce stability condition (5.11):

$$R(\Theta) = k \sum \sigma_r \left[\lambda_e(D(W_r - I) + (W_r - I)^T D)\right] \tag{5.12}$$

To use the regularization method (5.12), the equation $W_y^T = DW_u$ must have a diagonal positive definite solution D. This can be guaranteed in the training process by defining the RNN parameters in terms of D instead of W_y. In this way, the output equation for the ordinary RNN (3.1b) becomes $\boldsymbol{y} = (DW_u)^T \boldsymbol{x} + \boldsymbol{b}_y$ with the training parameters $\Theta_y = \{D, \boldsymbol{b}_y\}$.

Thus far, (5.12) has only been used for training 1-port models, since the requirement of the existence of D is easy to meet when $n_u = n_y = 1$. For 1-port models, W_u and W_y^T are all $n_x \times 1$ vectors; thus, as long as they share the same sign element-wise, the diagonal entries of D can be calculated from element-wise division between W_y^T and W_u. On the other hand, if there are two or more ports in the model, the parameters $W_u \in R^{n_x \times n_u}$ and $W_y^T \in R^{n_x \times n_y}$ are not vectors. In this case, the existence of D implies linear dependence of all vectors formed by element-wise division from the corresponding column vectors of W_y^T and W_u. That is a demanding constraint to be put on the two matrices. Further analysis of system (5.10) is warranted to derive a stability condition applicable to circuits with more than one input-output port.

For Hopfield networks, the nonlinear system (5.5) can be written as

$$\dot{\boldsymbol{x}} = h^{-1}\left[-\Lambda \boldsymbol{x} + W_r \tanh\left((I - W_u R W_y)\boldsymbol{x} + W_u \boldsymbol{V} - W_u R \boldsymbol{b}_y + \boldsymbol{b}_u\right)\right] \tag{5.13}$$

Unfortunately, this equation is neither in the form of a Hopfield network, nor any neural network structure with a known stability condition. Furthermore, a simple transform of states cannot convert the equation to a form for which a stability condition is known. Consequently, stability analysis of (5.13) needs to be done from scratch, in principle with the Lyapunov method. This analysis is beyond to the scope of this dissertation.

5.3 Test Case I: Nonlinear RLC Circuit

An experiment is designed and performed to test the effectiveness of the regularization method (5.12). In this experiment, a 1-port circuit comprised of a nonlinear resistor, an inductor and a capacitor is used as the test case. The circuit schematic is shown in Figure 5.1, and the nonlinear resistor has an I-V characteristic defined by the following function:

$$I = 4\tanh(V) + \frac{1}{4}V \tag{5.14}$$

ZIZO ordinary RNN models (3.3) with $n_x = 30$ are used for this experiment. A single set of training data is generated with the random PWL method, and used to train multiple models with or without the stability regularization. Each of the trained RNN models is converted to a Verilog-A model using the method introduced in Section 2.4.1.

The stability of the Verilog-A models is evaluated by transient simulation in which the model is excited by a step-voltage source through a linear resistor. For the simulation with each step-voltage and source resistance (V-R) combination, the system is considered stable if both the voltage and current responses of the model settle to a constant value. In contrast, if either of the voltage and current

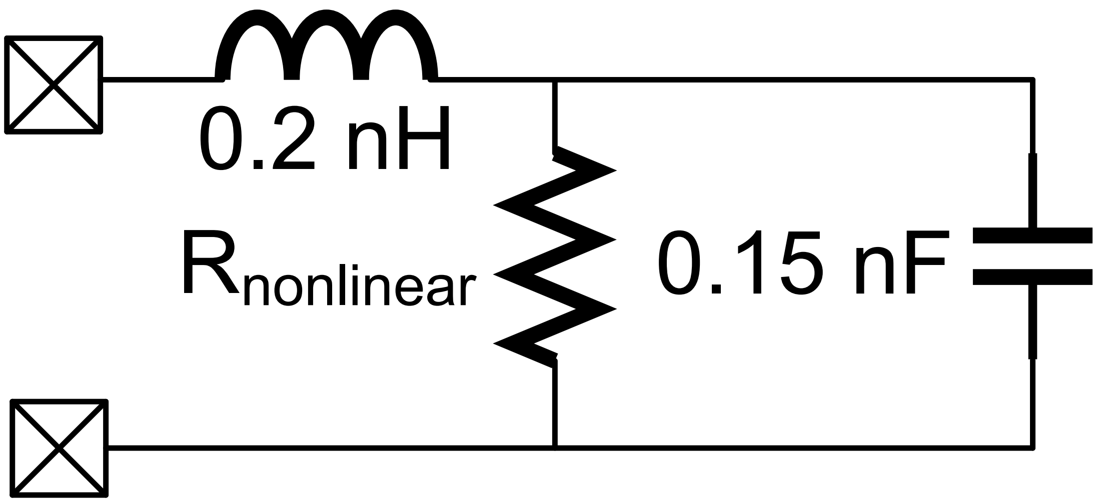


Figure 5.1: Netlist of the 1-port test circuit used for model stability analysis.

waveforms appears to be oscillating indefinitely, or exploding to infinity, the system is considered unstable. The result of all stability simulations can be visualized by calculating the maximum peak-to-peak variation in the voltage and current waveforms after a predefined time T, i.e.

$$V_{pp} = \max_{t>T} V(t) - \min_{t>T} V(t) \tag{5.15a}$$

$$I_{pp} = \max_{t>T} I(t) - \min_{t>T} I(t) \tag{5.15b}$$

and T is chosen to be long enough for the circuit to settle. An example of this visualization is shown in Figure 5.2, which shows the stability simulation result for the original circuit itself. Obviously, the original circuit is stable for all tested V-R combinations, since the simulated voltage and current variations all die out eventually. In fact, it can be proven that the original circuit is in theory ABST for port-wise resistive feedback. The proof is given in Appendix B.

Not surprisingly, if the stability regularization (5.11) is not used, it is possible for the trained RNN model to be unstable for specific V-R combinations, even if it appears to have good accuracy in all perspectives — training, validation and

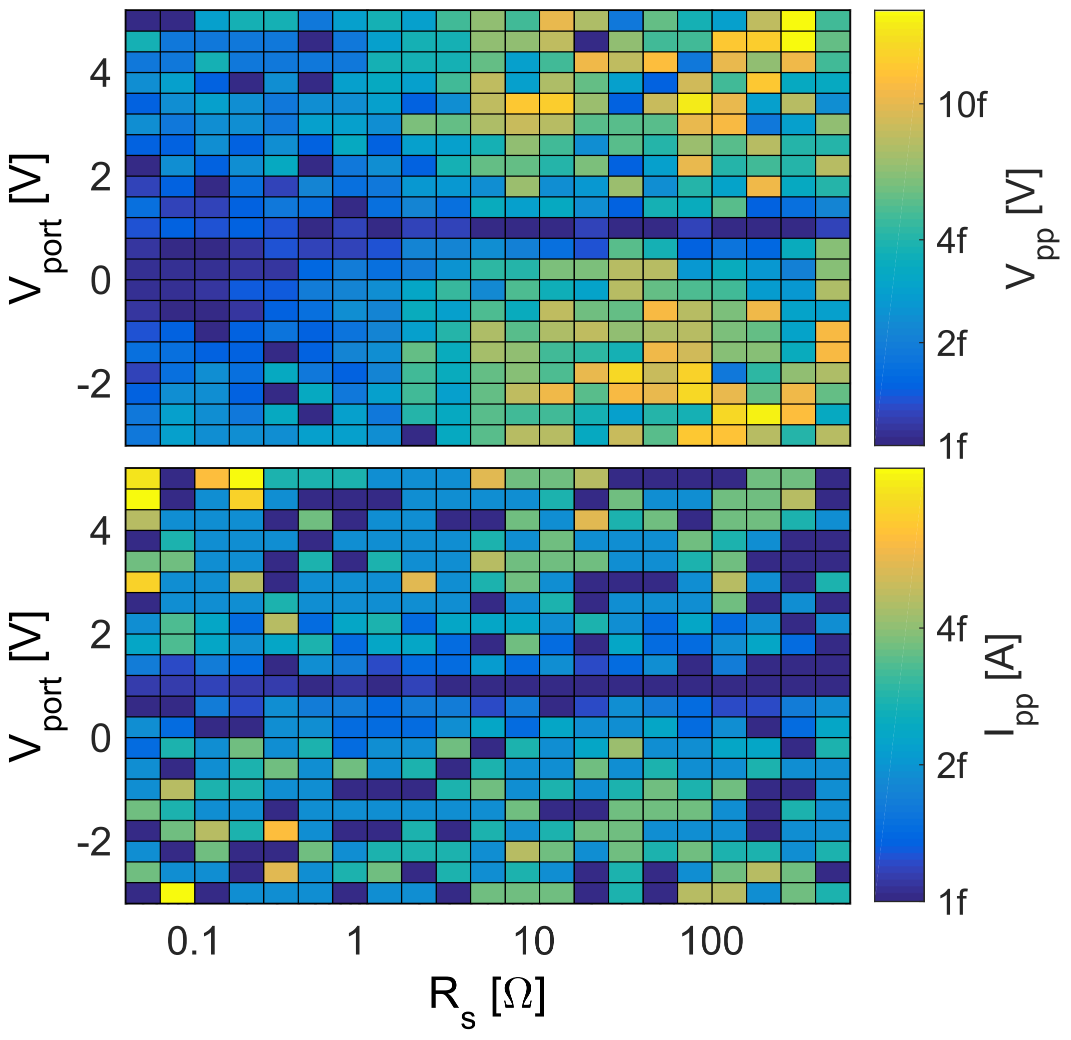


Figure 5.2: Stability simulation result for the original test circuit shown in Figure 5.1. Note the voltage scale on the color bar is in fV ($10^{-15}V$).

Verilog-A evaluation. An example transient simulation result is shown in Figure 5.3, highlighting the oscillatory behavior predicted by the RNN model. For this specific RNN model, the full stability simulation result is shown in Figure 5.4. Evidently, for some V-R combinations the system is shown to be oscillating at a significant amplitude, reflected with both the current and voltage waveforms. Notice that this oscillation may not be observed in the evaluation phase, since the model is still stable for most Thevenin sources. Therefore, it is essential to

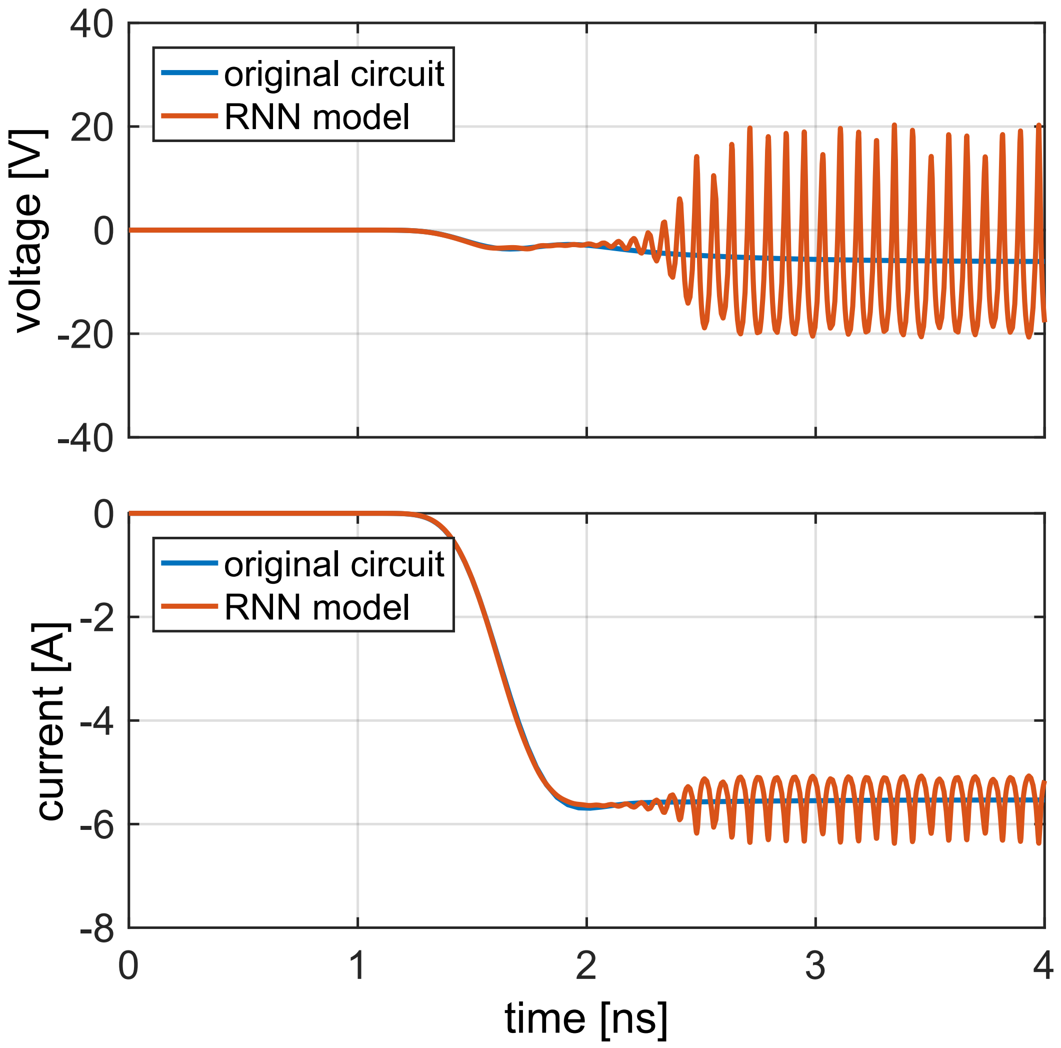


Figure 5.3: Transient simulation result for the test circuit and an RNN model, demonstrating the unstable behavior of the RNN.

run stability simulations if one desires to determine the stability of the model. An RNN model should be classified as unstable if *any* of the stability simulation results shows an unstable behavior.

To evaluate the effectiveness of the regularization method (5.12), 10 models are trained for the nonlinear RLC test circuit with or without the regularization. The weight parameter γ in the regularization (5.9) is chosen to be 0.05. Among the 10 models, 5 are trained with VICO configuration and 5 with CIVO. The

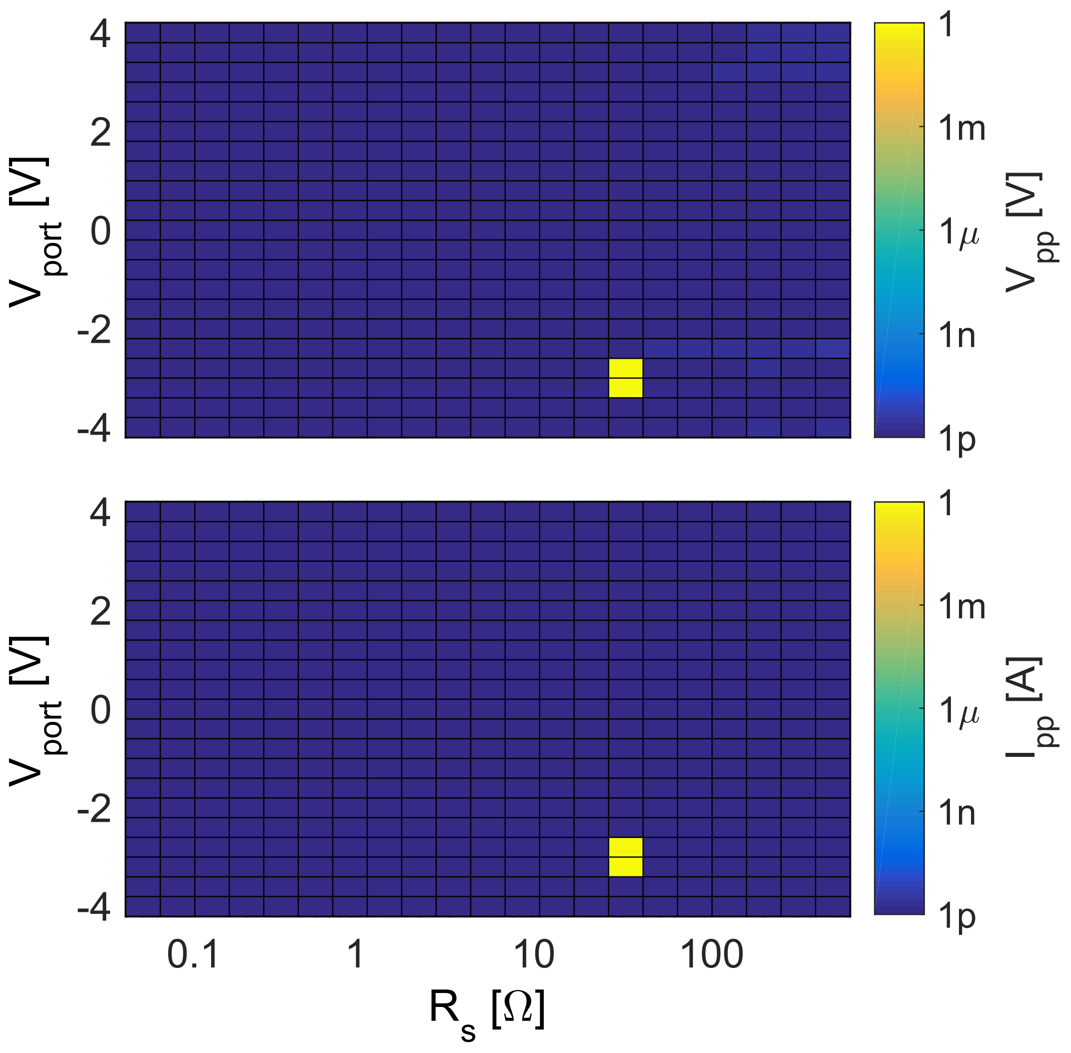


Figure 5.4: Stability simulation result for an RNN model of the nonlinear RLC stability test circuit.

only difference for each set of 5 models is the random initialization of the model parameters before training. Table 5.1 summarizes the accuracy and stability of the 10 RNN models trained without regularization. From Table 5.1, it appears that with a normal training approach without stability regularization, there is a non-zero probability that the trained model will be unstable.

For comparison, the accuracy and stability of the 10 models trained with stability regularization are listed in Table 5.2. In this case, all VICO models show

Table 5.1: Stability simulation result for ZIZO ordinary RNN models trained for the nonlinear RLC test case with no regularization.

Input	Train/Validation Error	Verilog-A Error	Stability
Voltage	3.6% / 3.9%	13.1% / 5.0%	Unstable
	0.8% / 0.8%	1.3% / 0.6%	Stable
	10.0% / 9.8%	7.9% / 3.9%	Stable
	2.3% / 2.1%	8.2% / 4.0%	Stable
	1.0% / 1.0%	2.5% / 1.3%	Stable
Current	4.0% / 4.2%	2.5% / 1.3%	Stable
	3.9% / 3.9%	2.2% / 1.1%	Stable
	4.1% / 4.0%	2.4% / 1.2%	Stable
	4.0% / 4.0%	2.7% / 1.3%	Stable
	10.2% / 7.1%	3.6% / 1.7%	Unstable

Table 5.2: Stability simulation result for ZIZO ordinary RNN models trained for the nonlinear RLC test case with regularization for port-wise resistive feedback stability.

Input	Train/Validation Error	Verilog-A Error	Stability
Voltage	16.4% / 16.4%	8.4% / 3.8%	Stable
	16.7% / 16.1%	7.6% / 3.4%	Stable
	15.9% / 15.6%	7.8% / 3.4%	Stable
	16.4% / 16.4%	7.9% / 3.6%	Stable
	16.5% / 16.7%	8.5% / 3.9%	Stable
Current	7.0% / 6.6%	1.9% / 0.8%	Stable
	5.3% / 5.3%	1.6% / 0.7%	Stable
	6.0% / 5.1%	1.7% / 0.7%	Stable
	5.1% / 5.1%	9.5% / 4.3%	Stable
	6.7% / 6.8%	1.8% / 0.8%	Stable

significant training and validation error, and the evaluation error, although lower than the training error, is still much higher than in the models trained without regularization. The increase in training error is expected, since regularization tends to constrain the model space. On the other hand, all 5 CIVO models give acceptable training and validation error only slightly higher than those of the no regularization case. Four of the models also show good evaluation error. Most importantly, all 10 models trained with the stability regularization are observed to be stable in the stability simulation for all V-R combinations, demonstrating the effectiveness of the regularization method. In conclusion, the stability regularization moderately diminishes the model accuracy but successfully guarantees the model stability. On the other hand, the difference in accuracy between the VICO and CIVO models is likely to be case-specific, and a rigorous physical explanation is hard to propose due to the black-box nature of the RNN model.

5.4 Test Case II: Two-port ESD Protection Circuit of an IO Pin

A second experiment on the stability of RNN models focuses on a 2-port ESD protection circuit for an IO cell. The schematic of the circuit is shown in Figure 5.5. Although for a circuit with two VICO or CIVO ports, no method is developed in this work to guarantee the RNN model stability, it will be demonstrated that the choice of input/output variables has an impact on the model stability.

Similar to the previous test case, random PWL stimuli are used for training data generation. The PWL source voltages and resistances are randomly sampled from distributions independent for each port. ZIZO ordinary RNN with $n_x = 30$ is chosen as the model structure. Five models are trained for both VICO and CIVO configurations, and for each configuration the only difference between the

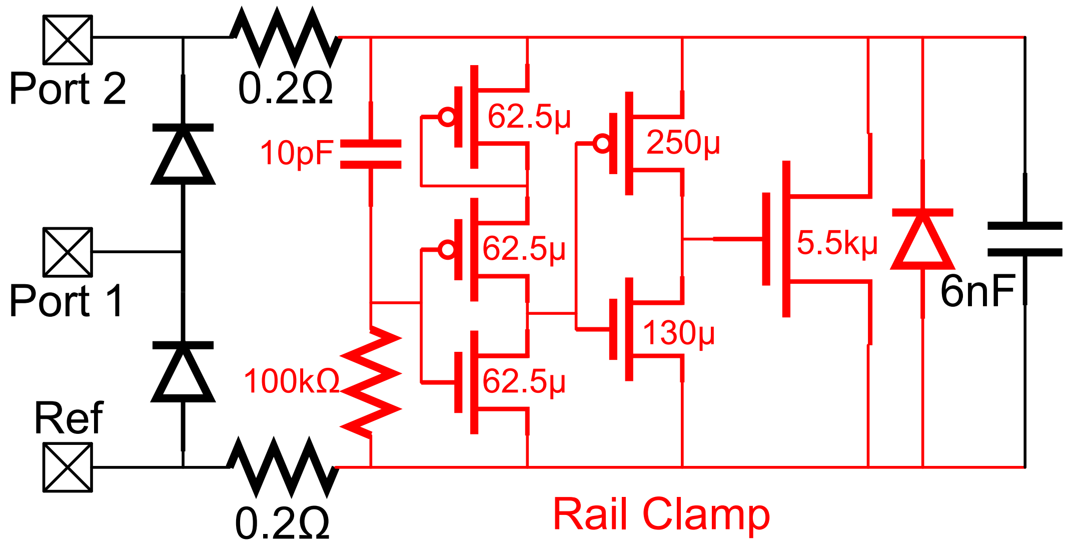


Figure 5.5: Schematics of the 2-port ESD protection circuit used for stability analysis. The rail clamp circuit in red is used in another test case in Section 6.1.1.

five models is the random initialization of the model parameters before training. The training and validation errors of the models are listed in the second column of Table 5.3. After training, each RNN model is implemented in Verilog-A as a 2-port behavioral model.

Rather than working directly on the 2-port RNN models, the evaluation of this circuit is carried out with the 1-port circuit created by shorting port 2 of the model. The reason of choosing this evaluation method is as follows: While being a test case for stability, there is no stability regularization applied during the training process. Hence, one cannot assume that the models will be ABST for port-wise resistive feedback. Therefore, the stability of the circuit should be evaluated with a more practical stability condition than the 2-port Thevenin stimuli method which tests the theoretical ABST condition. The 2-port Thevenin stimuli method itself is impractical due to the exceedingly large number of transient simulations needed for a full-factorial sweep of the voltages and resistances of both the Thevenin sources. Instead, the evaluation is carried out with the

Table 5.3: Stability simulation result for ZIZO ordinary RNN models for the 2-port ESD protection test case.

Port I/O	Train/Valid Error	Verilog-A Error V/I	Stability
VICO	3.8% / 3.8%	2.2% / 1.5%	Stable
	3.1% / 3.2%	2.0% / 1.5%	Unstable
	3.1% / 3.2%	2.3% / 1.7%	Unstable
	2.9% / 3.1%	2.7% / 1.9%	Unstable
	3.5% / 3.7%	1.4% / 1.0%	Unstable
CIVO	4.4% / 4.2%	1.9% / 1.4%	Stable
	4.2% / 4.0%	1.4% / 1.0%	Stable
	4.4% / 4.6%	0.9% / 0.7%	Stable
	4.6% / 4.6%	3.5% / 2.6%	Stable
	4.0% / 3.9%	1.5% / 1.1%	Stable

1-port circuit created by shorting port 2 of the model. The resultant 1-port circuit closely resembles the power-off ESD condition, in which the port 2 of the circuit is connected to a passive network with almost zero load impedance. The accuracy of the Verilog-A model is evaluated for the 1-port circuit using random PWL stimuli, and the stability is evaluated with Thevenin sources. The accuracy and stability of the 1-port is listed in columns 3-4 of Table 5.3.

In this test case, for both VICO and CIVO configurations, the training, validation and evaluation error for all five models are close to each other. Interestingly, the models trained with VICO configuration are unstable with a high probability, while the CIVO models are uniformly stable. This result offers a useful hint for real life training tasks, namely that it is advisable to change the input/output configurations of the model when stable models appear to be hard to achieve.

CHAPTER 6

EVALUATION OF MODELING METHODOLOGY WITH ADDITIONAL TEST CASES

In this chapter, the RNN modeling methodology is evaluated with circuits used in practical circuit designs. Selected circuits are modeled using RNNs introduced in Chapter 2 and training data generated with methods introduced in Section 2.2. It will be demonstrated that although some potential problems for RNN are exposed in Chapters 4 and 5, it in general creates an accurate model for many real life circuits if the training is done correctly. On the other hand, it will also be demonstrated that RNN models can deviate from accurate prediction when used in conditions for which they are not trained.

6.1 ESD Test Cases

As is discussed in Section 1.1, ESD protection circuits are important test cases for the RNN modeling methodology. This section shows some case studies for creating RNN models for ESD protection circuits.

6.1.1 Simple ESD Circuits

For the first ESD test set, the RNN is used to model two relatively simple functional circuits for ESD protection purposes. The two circuits being modeled are a) a single SPICE diode, and b) an active rail clamp circuit [47]. The schematic of the rail clamp circuit is shown in Figure 5.5 (the red portion), and the transistor models are BSIM4v4 with parameters defined for a commercial 130 nm CMOS technology. The ZIZO ordinary RNN (3.3) is used as the model structure. The training data are generated with circuit simulation, but the stimuli are chosen to be measurement-compatible, as is introduced in Figure 2.3. The trained models are implemented in Verilog-A and evaluated with stimuli different from those used in training data generation. The first evaluation stimulus is a random PWL voltage source with a $1\,\Omega$ output resistance. The second stimulus corresponds to an IEC 61000-4-2 ESD tester [3] whose model netlist is shown in Figure 6.1. Since both evaluation stimuli have a longer time-span than the training samples, it can be concluded that the model well replicates the internal dynamics of the circuit if the model accurately predicts the transient response of the circuit for all evaluation simulations.

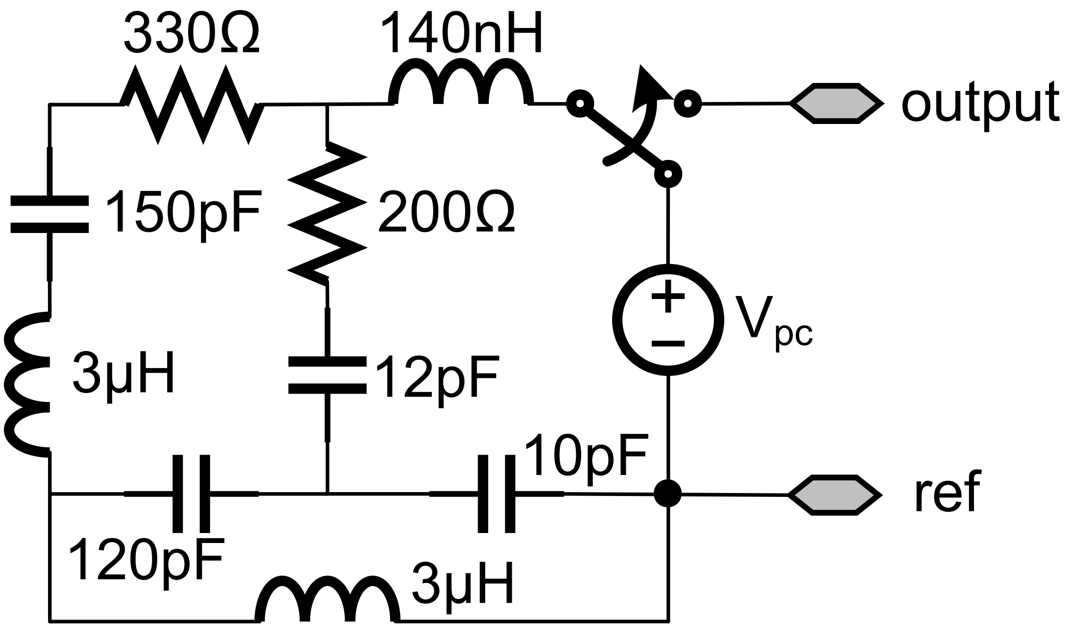


Figure 6.1: Schematic of the IEC 61000-4-2 ESD tester model. V_{pc} denotes the precharge voltage, and the relay needs to be actuated to deliver the discharge.

Figures 6.2 and 6.3 compare the simulation results obtained for the SPICE diode with original circuit model and the Verilog-A RNN model for the two evaluation stimuli. The same comparison for the rail clamp circuit is shown in

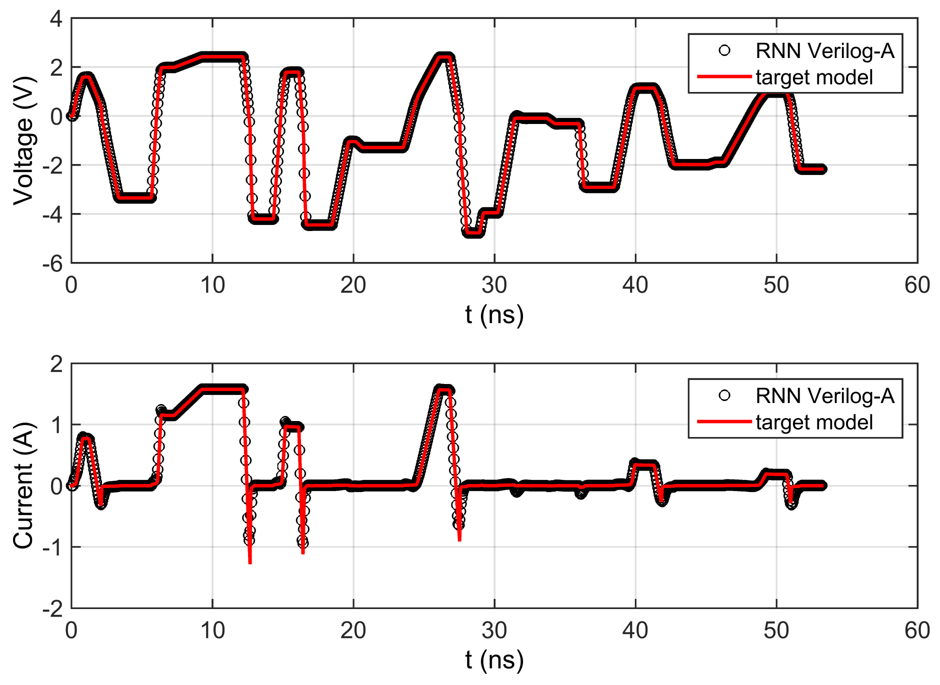


Figure 6.2: Simulation result for diode with PWL stimulus.

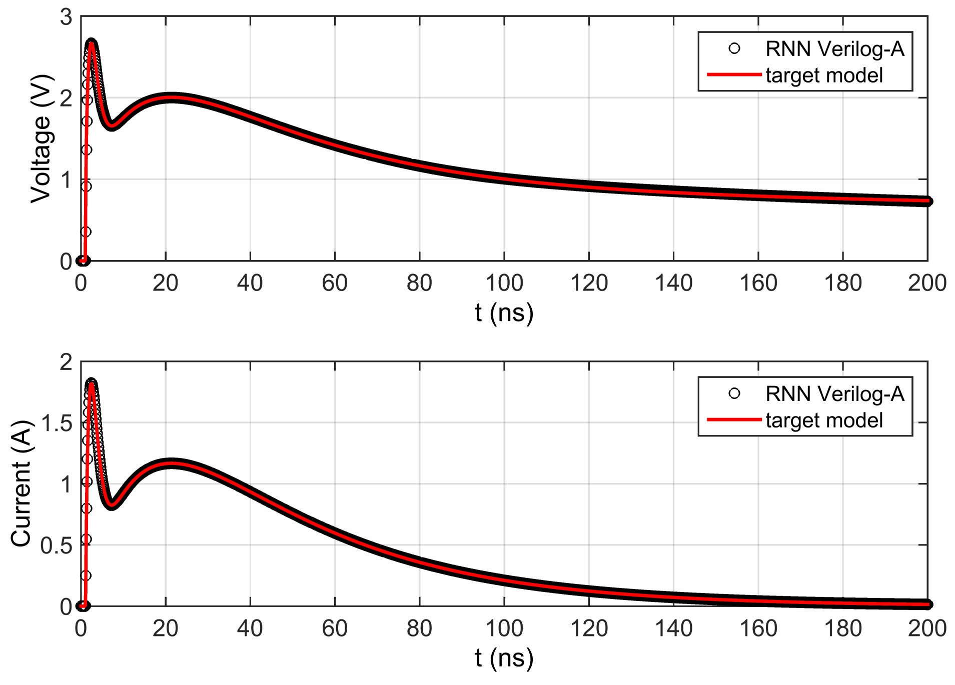


Figure 6.3: Simulation result for diode with IEC stimulus.

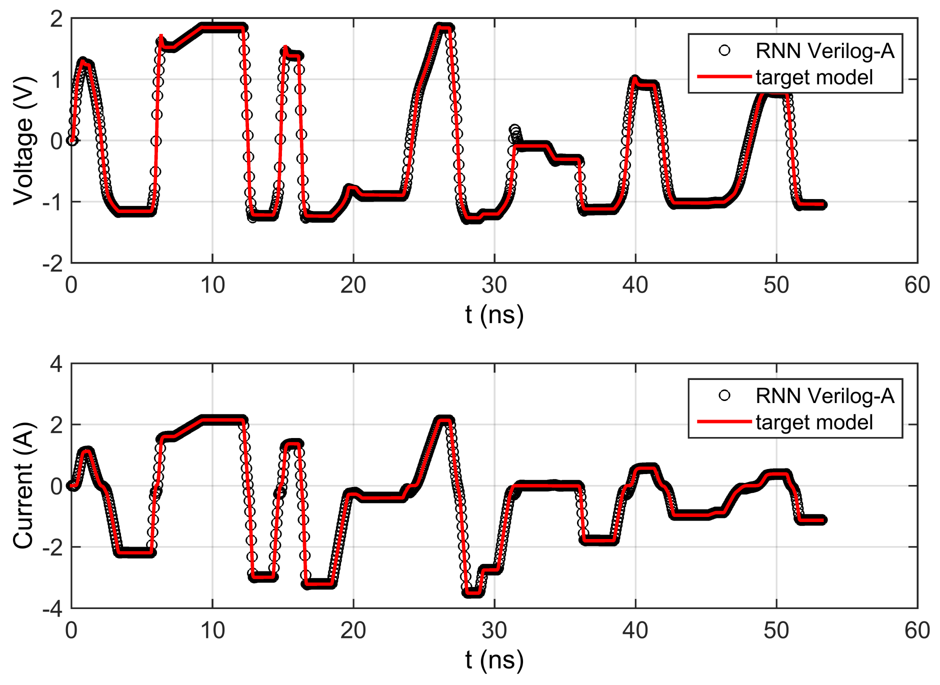


Figure 6.4: Simulation result for rail clamp with PWL stimulus.

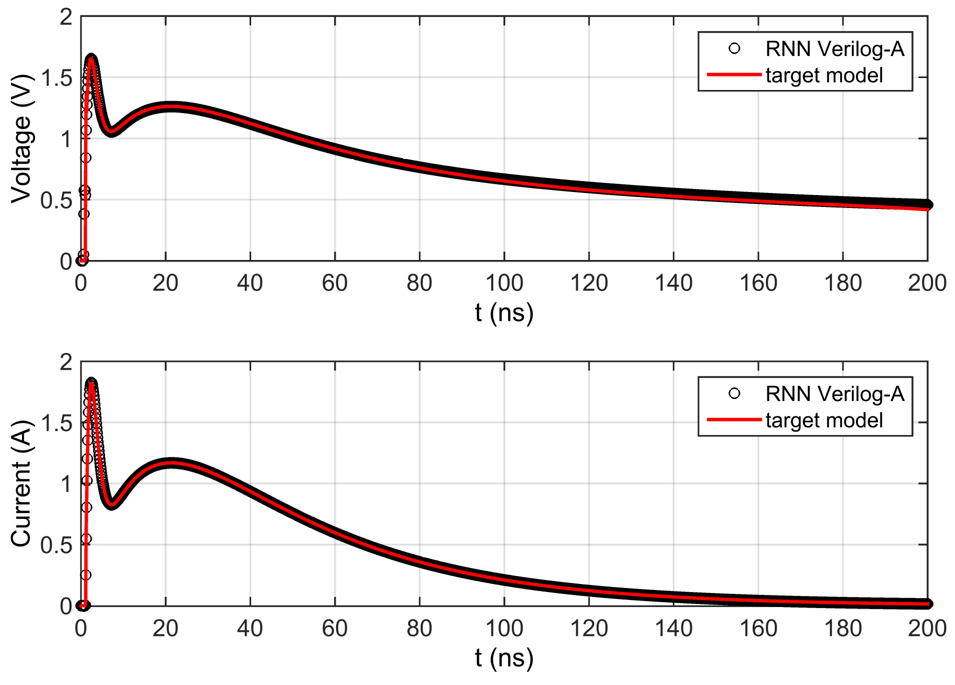


Figure 6.5: Simulation result for rail clamp with IEC stimulus.

Table 6.1: Error of RNN model for the simple test cases.

Case	Stimuli	Voltage/Current Error
Diode	Random PWL	0.64% / 1.66%
Diode	IEC	0.55% / 0.15%
Rail Clamp	Random PWL	1.39% / 0.77%
Rail Clamp	IEC	1.91% / 0.20%

Figures 6.4 and 6.5. The relative errors are calculated with a slightly different formula than (3.21):

$$E_r = \frac{\sqrt{\frac{1}{L}\sum_{i=1}^{L} \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|^2}}{\max \boldsymbol{y}_i - \min \boldsymbol{y}_i} \tag{6.1}$$

The calculated errors are listed in Table 6.1. Note that with (6.1), the calculated relative error will be smaller than (3.21). However, it appears that in all cases the RNN models have an error of less than 2%, indicating that the RNN models accurately predict the transient response of both test circuits.

6.1.2 Full-chip ESD Protection Network

In the second test case, a more complicated circuit — a full-chip ESD protection network — is used as the modeling target. The circuit schematic is shown in Figure 6.6, with the voltage sources $\mathrm{V_{SIO}}$ and $\mathrm{V_{Score}}$ both set to zero to emulate the power-off condition. Although the circuit contains many elements, it is a 1-port with the IO pin as positive terminal and ground as negative terminal. The inductances in this circuit are identified to be large enough to cause long-term memory related issue, i.e. the vanishing gradient problem. Therefore, the RNN most robust against the vanishing gradient problem, the ZIZO GRU, is chosen to be the model structure. The hidden states vector size of the ZIZO GRU is set to $n_x = 30$. On the other hand, notice that an inductor appears in series with the input terminal, the IO pin. Since the inductance is known,

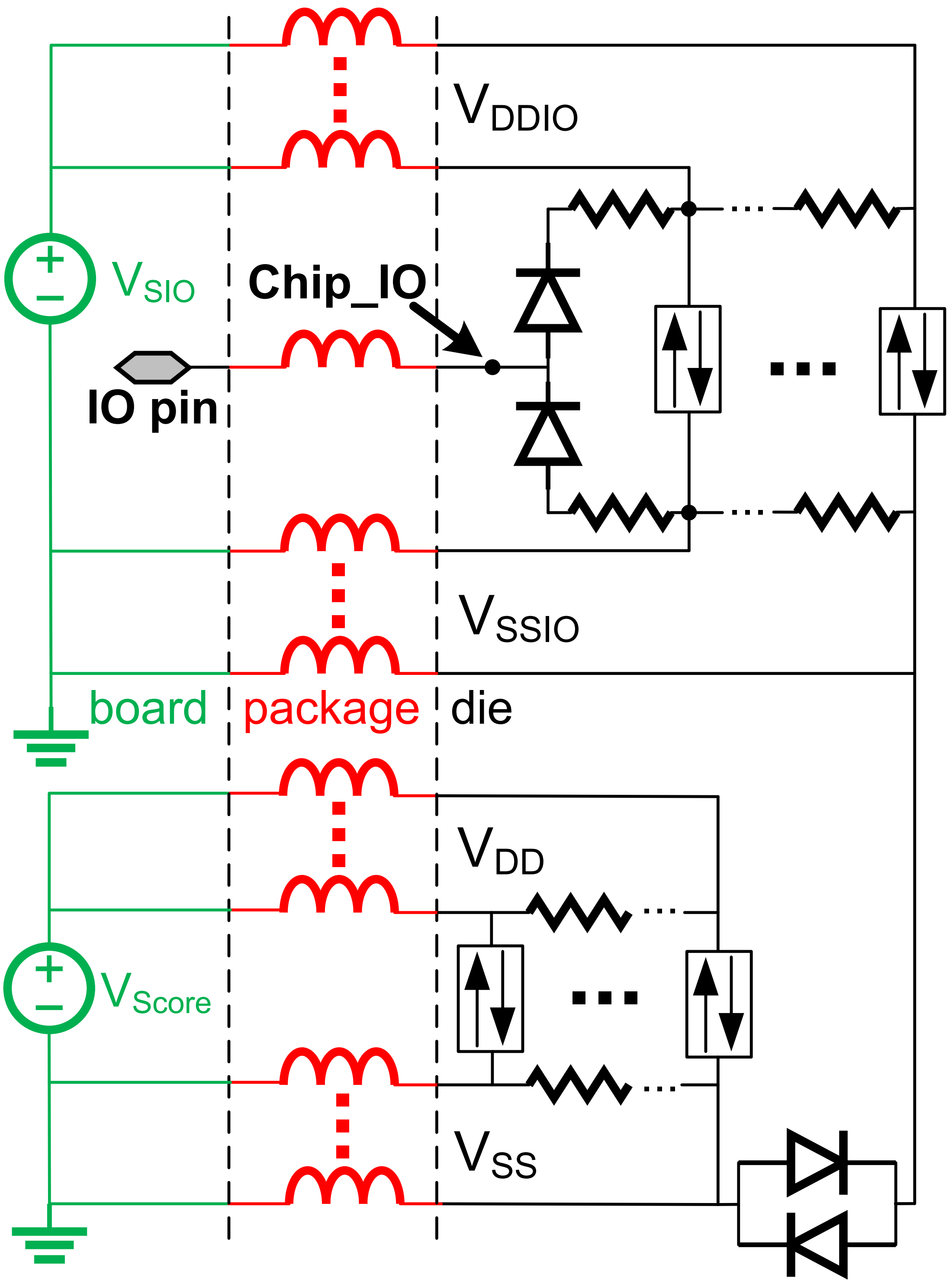


Figure 6.6: Schematic of the ESD protection network test case. The box with bidirectional arrow represents active rail clamp plus decoupling capacitor, with the VDDIO domain version shown in Figure 5.5 and the VDD domain version sharing the same structure with different sizing.

Table 6.2: Model Accuracy of ZIZO GRU model of the ESD protection network, modeled from IO pin.

Stimuli	Error
Training	Training 1.4% / Validation 1.6%
Random PWL	Voltage 0.89% / Current 1.80%
IEC	Voltage 45.06% / Current 0.10%

it is possible to exclude it from the RNN model by modeling the ESD network as a 1-port between the node labeled "Chip_IO" and ground. The full circuit model will consist of the RNN model in series with the IO pin inductance. In this way, the total amount of memory in the RNN model is reduced, and this may alleviate the vanishing gradient problem. In principle, any circuit element strictly in series with a port, i.e. current flowing through the element equal to the current flowing into the port at all times, can be excluded from the RNN model if needed. However, all the other inductances shown in Figure 6.6 do not satisfy this criterion as each of them is in parallel with another inductor connected to the same power/ground net, lumped from multiple power/ground pins.

The two 1-port realizations of the ESD network, one from IO pin to ground and the other from Chip_IO to ground, are treated as two completely independent test cases and trained separately using the same ZIZO GRU model structure. For both circuits, training data are generated using random PWL stimuli. Similar to the test cases shown in Section 6.1.1, the trained Verilog-A RNN models are evaluated with randomly generated PWL stimuli and the IEC 61000-4-2 tester model.

The accuracy of the ZIZO GRU model trained for the ESD network with IO pin as the input is listed in Table 6.2. The transient simulation results with the two evaluation stimuli are plotted in Figures 6.7 and 6.8. It appeared that the GRU is capable of handling the long-term memory of the system, showing

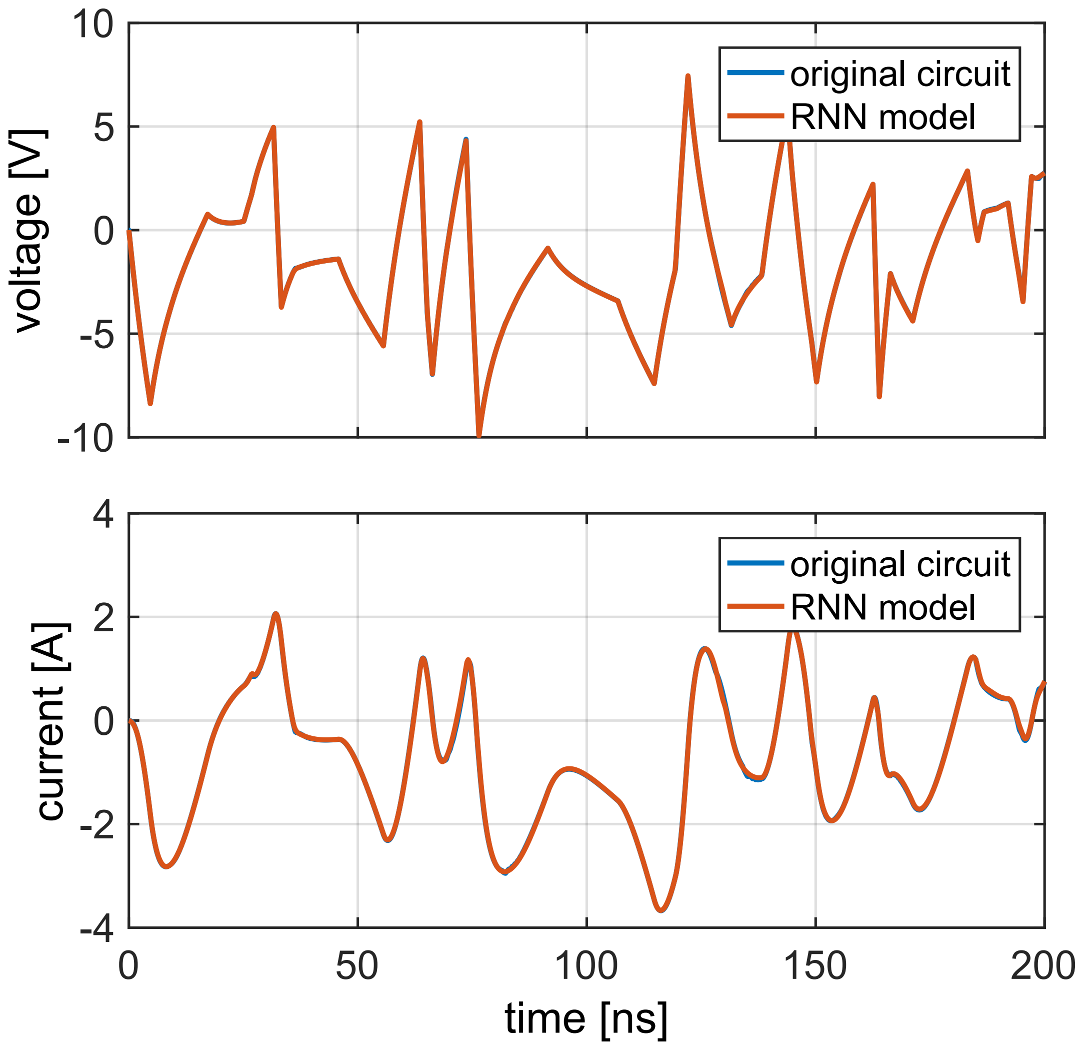


Figure 6.7: Verilog-A GRU simulation result with random PWL stimuli for the ESD network model with input at IO pin.

excellent training and validation errors. The Verilog-A model also shows low error for random PWL evaluation. However, in the transient simulation with IEC stimuli, the Verilog-A RNN model prediction deviates from the original circuit by a significant margin. From Figure 6.8, it is evident that most of this error comes from the initial few nanoseconds in the simulation voltage waveform. A zoomed-in view of this voltage waveform is presented in Figure 6.9. Clearly, the Verilog-A RNN model is incapable of replicating the very fast oscillatory response of the

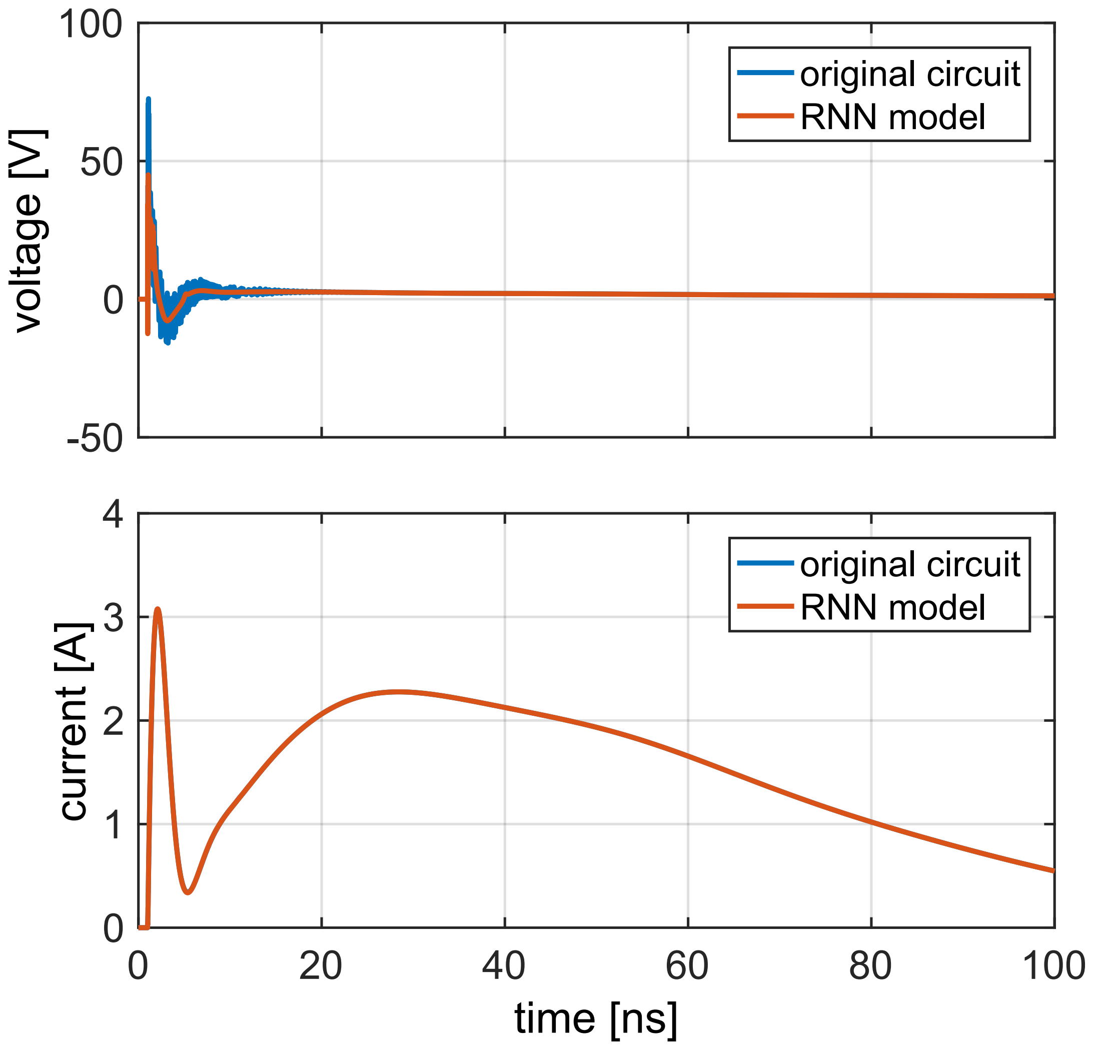


Figure 6.8: Verilog-A GRU simulation result with IEC stimuli for the ESD network model with input at IO pin.

original circuit. The decreased predictability of the RNN for very fast oscillatory waveforms is attributed to a lack of high-frequency information in the training data. In fact, in the IEC 61000-4-2 standard, the specified current rise time for the first current peak is between 600 and 1000 ps. Using this information, it is determined for training data generation that the fastest τ_i parameter selected for the random PWL waveform (2.2) is 500 ps so that the fastest current pulse in the IEC waveform can be covered; and the time step used for the time series

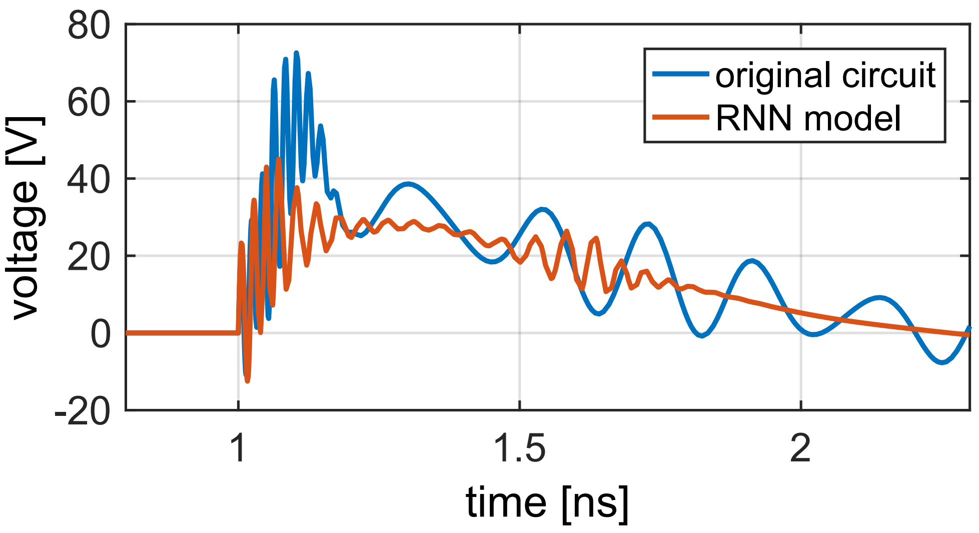


Figure 6.9: Zoomed in view of the voltage waveform in Figure 6.8.

interpolation is $h = 50\,\text{ps}$ so that sufficient temporal resolution can be achieved for those fast pulses. Therefore, the accuracy of the RNN model is expected to collapse if the stimulus applied has a rise time much faster than 500 ps, or frequency much higher than 1 GHz. From Figure 6.9, the initial oscillation in the voltage waveform has a frequency of about 50 GHz, while the oscillation frequency at 2 ns is about 5 GHz. Those high-frequency oscillations are not seen by the RNN during the training process. Therefore, it would actually be surprising if the RNN trained without such high-frequency information would be able to accurately predict the response.

In theory, the reduced model accuracy for high-frequency waveforms can be mitigated by using faster stimuli during training data generation and shorter time step for interpolation. However, in this case the change in time step would be too aggressive since the oscillation frequency can be as high as 50 GHz. Attempting to match this in training data generation would lead to severe vanishing gradient problem and drastic increase in the training time.

For the test case in which the input of the model is Chip_IO, the evaluation

Table 6.3: Model accuracy of ZIZO GRU model of the ESD protection network, modeled from Chip_IO.

Stimuli	Error
Training	Training 1.6% / Validation 2.4%
Random PWL	Voltage 1.62% / Current 2.44%
IEC @ IO pin	Voltage 11.12% / Current 0.05%
IEC @ Chip_IO	Voltage 36.39% / Current 0.05%

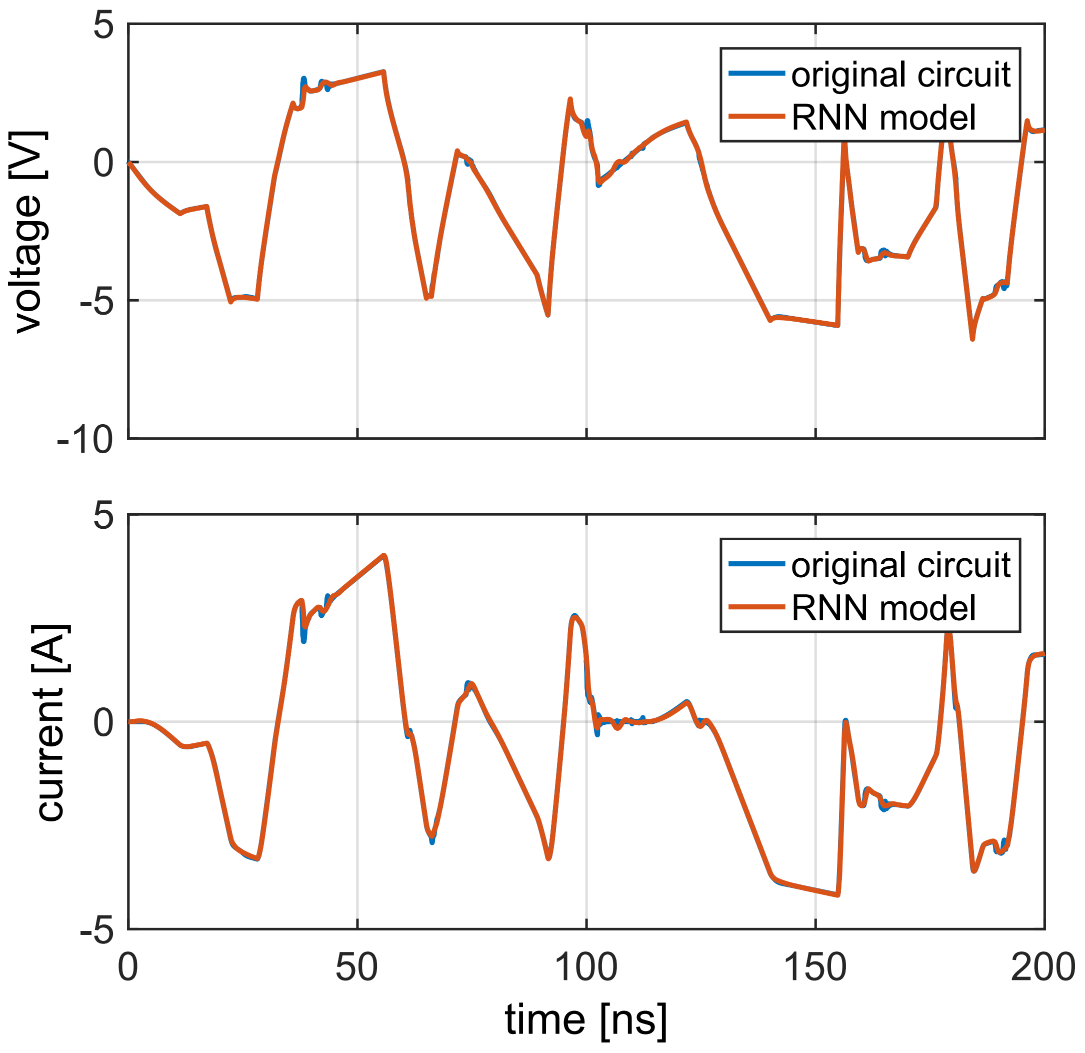


Figure 6.10: Verilog-A GRU simulation result with random PWL stimuli for the ESD network model with input at Chip_IO.

is done in a slightly different way. The evaluation with random PWL stimuli is applied to the RNN model as usual. The IEC tester model, however, is not

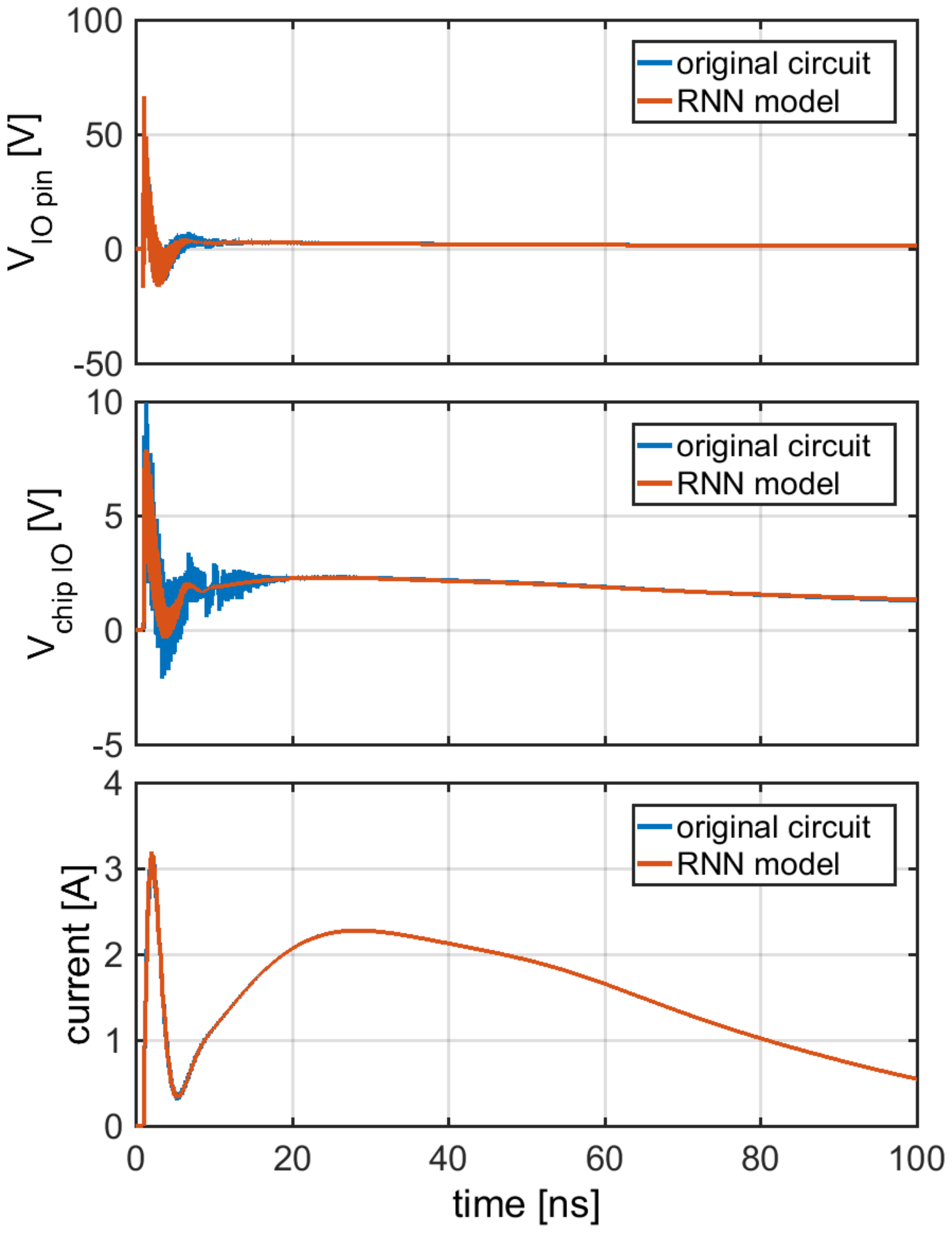


Figure 6.11: Verilog-A GRU simulation result with random PWL stimuli for the ESD network model with input at Chip_IO.

connected directly to the RNN model, as the evaluation with IEC tester resembles a real world measurement case, and it would be unrealistic to directly measure

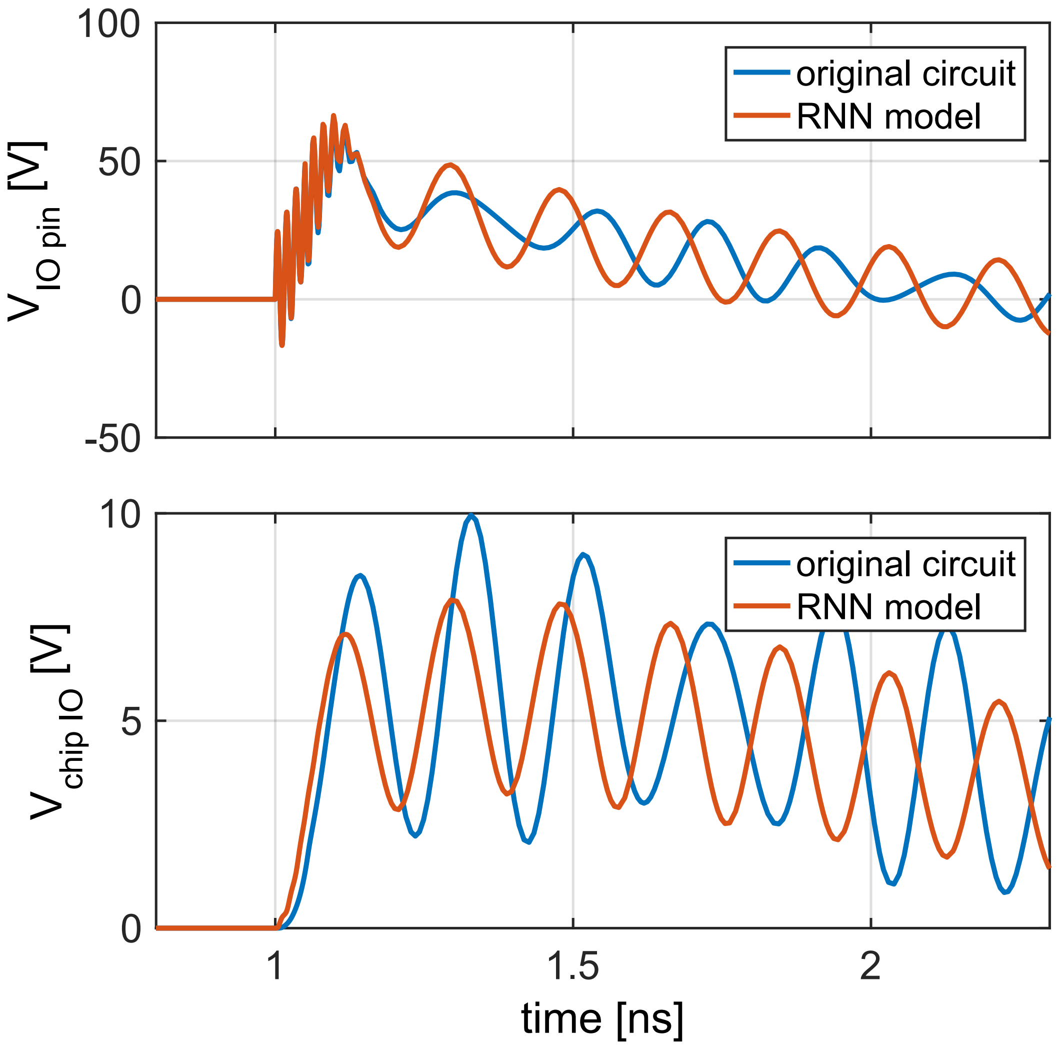


Figure 6.12: Zoomed in view of the voltage waveforms in Figure 6.11.

Chip_IO. Instead, the IEC tester is connected to the full network model, i.e. RNN model in series with the IO pin inductance. Therefore, for the evaluation with IEC tester, voltage and current waveforms at both the IO pin and Chip_IO are compared with the simulation result using the original circuit, and errors are also quantified for both nodes.

Table 6.3 summarizes all the model accuracy figures-of-merit for the Chip_IO test case, and the comparison between transient simulation results with the original circuit and the Verilog-A RNN model is shown in Figures 6.10 and 6.11.

Again, the training, validation, and random PWL evaluation error for the model are all small, while the IEC evaluation shows large error in the predicted voltage waveform. The error is still concentrated in the first few nanoseconds of the waveform. In Figure 6.12, zoomed-in views of the simulated voltage waveforms at the IO pin and Chip_IO are both plotted. In this case, for the voltage waveform at the IO pin, the 50 GHz oscillation is well matched while the 5 GHz oscillation is not. Circuit analysis shows that the 50 GHz oscillation is mostly contributed by the inductor between the IO pin and Chip_IO node. By excluding this inductor from the RNN model, both simulations (RNN and original circuit) derive this oscillation from lumped circuit elements, which are identical in the first place. The 5 GHz oscillation is caused by the circuit being modeled, since it also appears in the waveform at Chip_IO. Since the random PWL stimuli used for training data generation have a fastest rising edge of 500 ps, the RNN prediction of this 5 GHz oscillation is expected to be inaccurate. In Figure 6.12, the RNN demonstrates some generalization capability by predicting a fairly close amplitude and frequency for the 5 GHz oscillation, but the phase error becomes large very soon.

In conclusion, for the two full-chip ESD protection network test cases, the RNN (ZIZO GRU) model accurately replicates the behavior of the original circuit for signal frequencies that appeared in the training data. To improve the RNN accuracy for faster signal, the training data has to be regenerated with faster stimuli, and interpolated to time series with a shorter time step. This change will in general aggravate the vanishing gradient problem and lead to worse training accuracy. Compared with aggressively scaling the time step, a better approach would be to investigate the circuit being modeled first and, if possible, exclude the circuit elements causing the highest frequency response from the RNN model.

6.2 Other Behavioral Modeling Tasks

In this section, some circuits whose purpose is other than ESD protection are modeled using RNN. This describes the majority of integrated circuit blocks.

6.2.1 CTLE Circuit

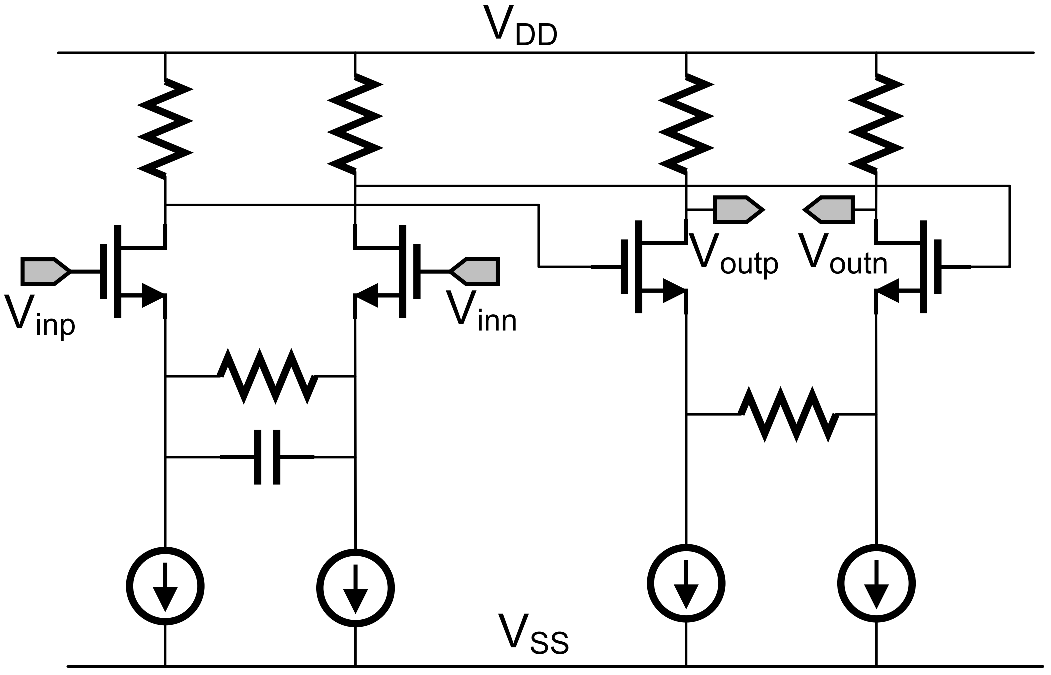


Figure 6.13: Schematic of the CTLE circuit.

A continuous-time linear equalizer (CTLE) circuit is modeled using an ordinary RNN. The schematic of the circuit is shown in Figure 6.13. The CTLE has differential input and output pairs. Assuming almost-ideal source and almost-open load, the RNN structure is chosen to have two VINO ports and two NIVO ports. The hidden state vector length n_x is arbitrarily chosen to be 20 because the behavioral complexity of the circuit is expected to be low. The regularization term (5.8) is applied to enforce the stability of the model and the regularization weight γ in (5.9) is chosen to be 0.05.

Table 6.4: Model Accuracy of RNN model of the CTLE circuit.

Stimuli	Error
Training	Training 0.6% / Validation 0.6%
Evaluation	V_{outp} 1.87% / V_{outn} 1.91% V_{dm} 11.70% / V_{cm} 1.00%

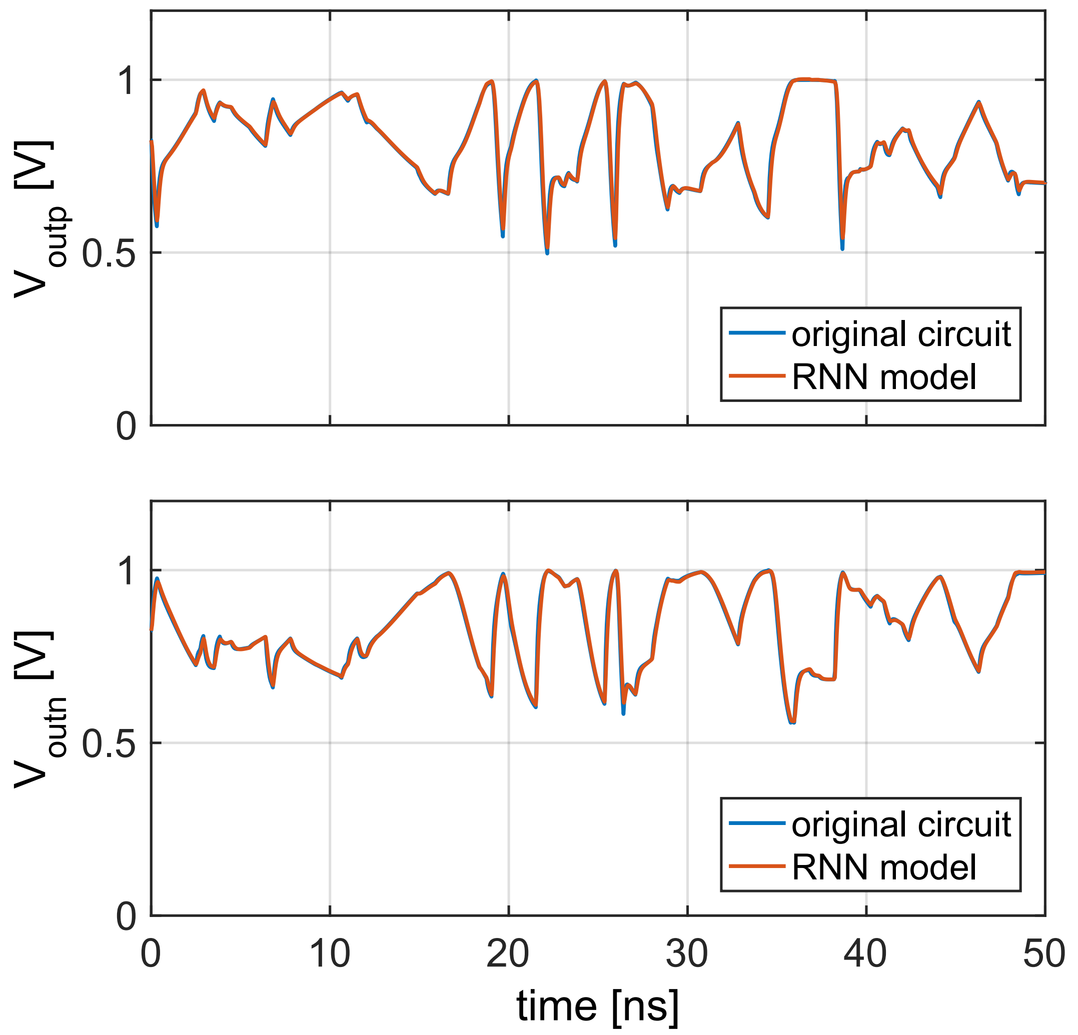


Figure 6.14: Evaluation simulation result of the CTLE test case, presented in terms of voltage waveforms at the two outputs.

Training data is generated for the CTLE circuit with a slightly modified random PWL stimuli. In this case, the PWL waveforms are not generated for the

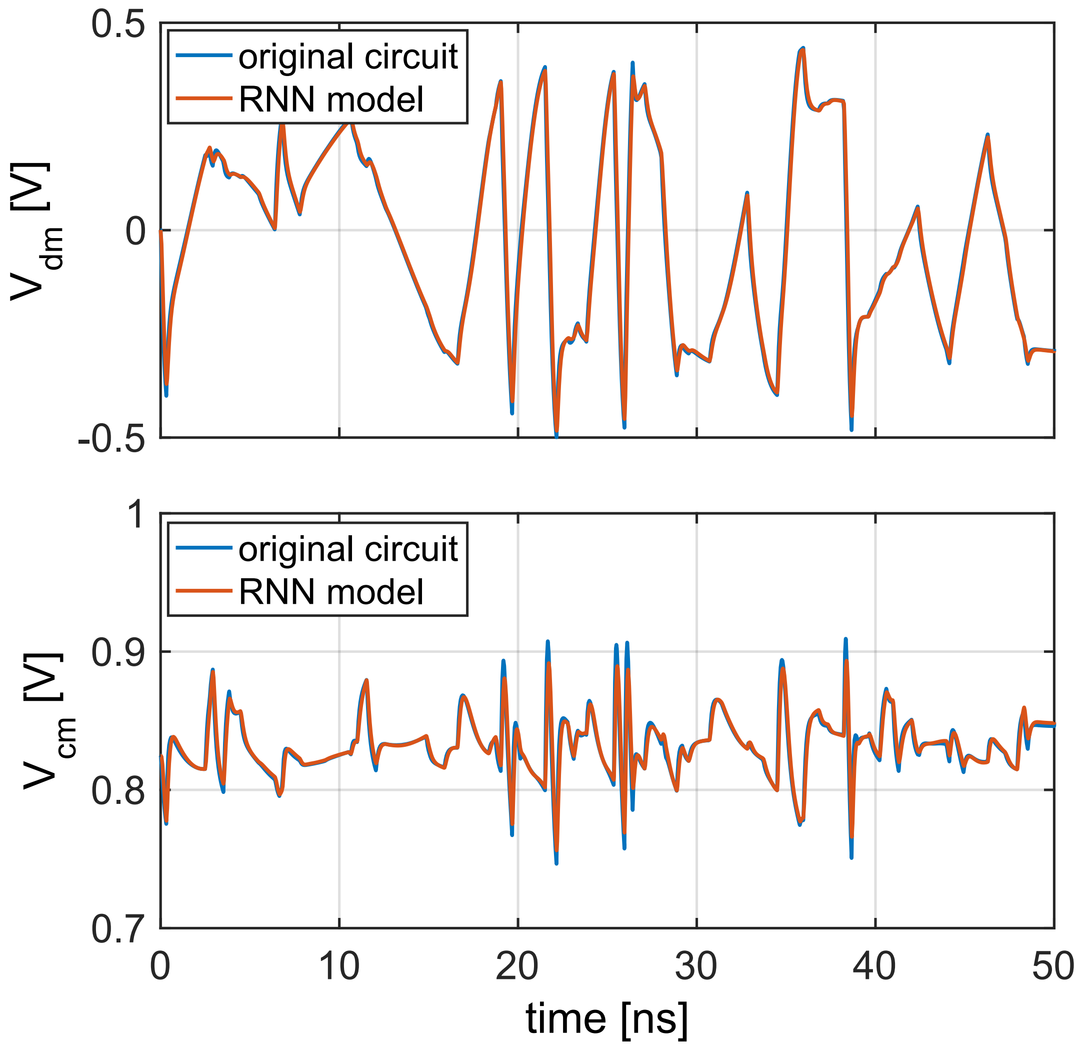


Figure 6.15: Evaluation simulation result of the CTLE test case, presented in terms of differential-model and common-mode voltage waveforms.

voltage at the two inputs, but rather for the common-mode and differential-mode voltage between the two inputs. The rationale for this data generation method is the fact that the common-mode voltage applied to the inputs of the circuit is relatively constant. If full-range random PWL waveforms are independently applied to both inputs, the common-mode voltage waveform appearing at the inputs would become unrealistically large. Instead, the differential-mode waveforms cover the full range of expected (allowed) amplitudes, while the common-mode waveforms only deviate slightly from the nominal value. The voltages at the two input ports are finally calculated from the common-mode and differential-mode voltages.

As usual, the training data is interpolated to time series for training, and the trained model is implemented in Verilog-A for evaluation. Table 6.4 summarizes the model accuracy, while Figures 6.14 and 6.15 compare the simulated output waveforms in evaluation for the original circuit and the Verilog-A RNN model. Based on these results, it can be concluded that the RNN model provides a good representation of the CTLE circuit.

6.2.2 An Encrypted Circuit Netlist

The last test case is an unknown circuit described by an encrypted netlist. None of the design details were disclosed, other than the fact that the circuit contains more than 1000 transistors, each of which is described by a BSIM4 model. The only information provided is a list of the terminals and the expected stimuli. The circuit has 12 terminals: 3 terminals are connected to constant voltage or current sources; 7 terminals are input ports, subject to an arbitrary input signal; 2 terminals are outputs. At first glance, it might seem highly challenging to generate sufficient training data for black-box modeling of this circuit; generating training data for a 7-input system with all inputs independently subject to arbitrary waveforms would consume a prohibitive amount of computational power. However, for this circuit, the types of stimuli applied to the 7 inputs are highly limited. The inputs can only stay at two static voltage levels, 0 V and 5 V, except when a level transition happens. The level transitions occur only at 1 ms intervals, synchronized for all input terminals, and the rising and falling time for the transition is always 1 ns. In this way, every input can be represented by

Table 6.5: Model accuracy of RNN model of the unknown encrypted circuit.

Stimuli	Error
Training	Training 2.0% / Validation 2.0%
Evaluation	Output-1 0.72% / Output-2 0.74%

Table 6.6: Simulation speed for the RNN model of the unknown encrypted circuit.

Netlist	Simulation time
Original circuit	254.5 s
RNN model	5.7 s

a string of 0's for the low voltage level and 1's for the high voltage level.

It can be shown that the total number of input state transitions is $(2^7)^2 = 16384$, and it is possible to create training stimuli that cover all the level transitions exhaustively. In practice, the training stimuli are chosen to be random strings of 0's and 1's at all inputs. The training data are generated by transient circuit simulation and the simulated waveforms are interpolated to time series with $h = 10\ \mu$s time step. It might seem odd to use a time step greater than the rise time of the input signal, but in this case the shape of the rising edge is not important information for predicting the circuit behavior. In addition, it is permissible to use a large time step since the response at the circuit outputs is slow.

The circuit is modeled as an RNN with $n_u = 7$, $n_y = 2$, and $n_x = 30$. The trained RNN is implemented in Verilog-A, and evaluated for stimuli that are dual-level voltage waveforms of longer duration than the waveforms in the training set. Table 6.5 summarizes the model accuracy, while Figure 6.16 compares simulated output waveforms for the original circuit and the RNN model. From those results, it can be concluded that the model accuracy is good.

Significantly, for this test case, the RNN model runs much faster than the

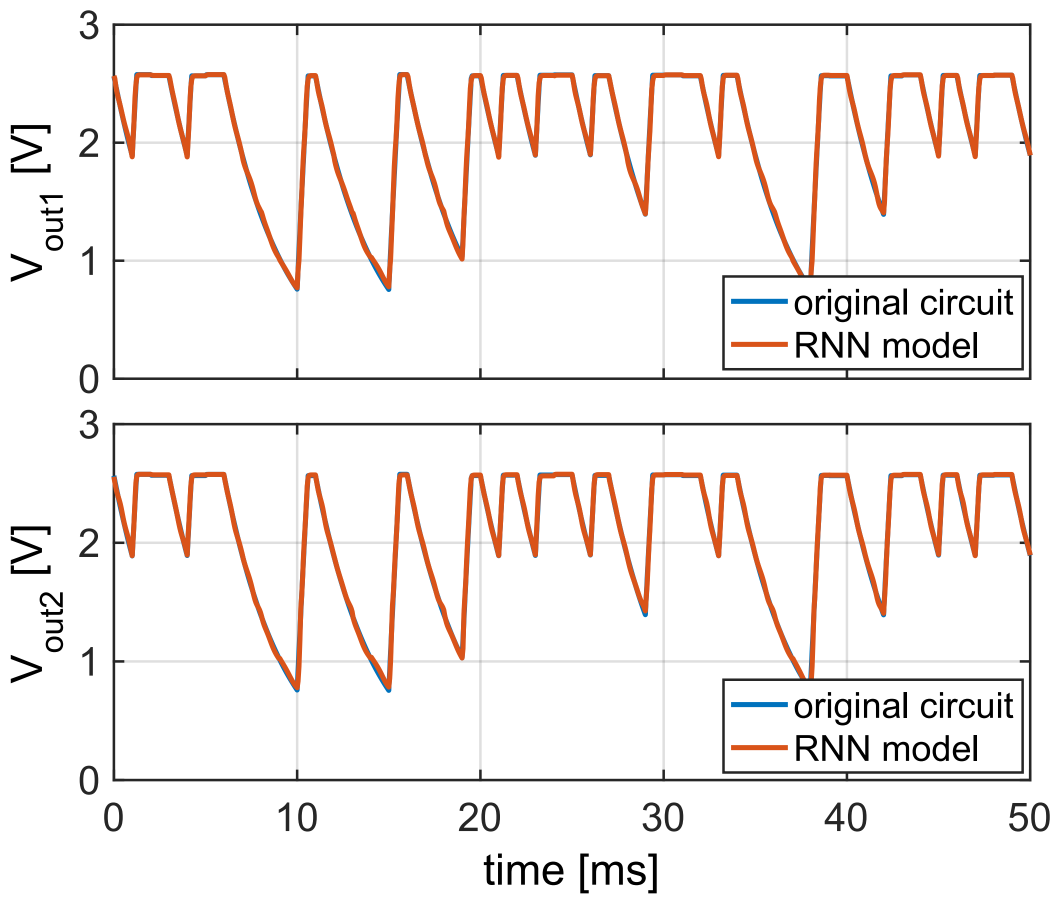


Figure 6.16: Evaluation simulation result for the unknown encrypted circuit netlist.

original circuit model in circuit simulation. Table 6.6 compares the CPU time consumption of the transient simulations. The RNN model achieves a more than 40x acceleration. In this case, the original circuit netlist contains more than 1000 transistors all modeled with BSIM. It is not surprising that such a large netlist takes a long time to simulate. On the other hand, the speed of the RNN model only depends on the model structure itself, i.e. the type of the RNN and the number of inputs, outputs, and hidden states. Therefore, if the structure of the RNN is kept the same, the acceleration provided by the RNN will increase with the complexity of the original circuit.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

It has been demonstrated in this work that RNNs can accurately model a variety of electrical circuits when used correctly. Especially, an RNN model is expected to well replicate the dynamic behavior of the circuit being modeled for the types of stimuli that are covered by the training data. On the other hand, certain inherent problems are identified for RNN model structures, and from them guidelines for using RNN in circuit modeling can be derived. The vanishing gradient problem hampers the training process especially for systems with long-term memory. The problem can be alleviated by increasing the time step of the training data through re-interpolation, but the model accuracy for fast stimuli will deteriorate as a trade-off. Thus, for a circuit with a large ratio between the length of its memory and the time constant of the fastest stimuli it encounters in expected operation conditions, it is advisable to use RNN structures more robust to the vanishing gradient problem. Among the three RNN structures introduced in this work, the ordinary RNN is the least robust to vanishing gradient; the Hopfield network is slightly better; the GRU is by far the best model for systems with long-term memory. However, even the GRU fails to train for systems with a long pure delay, and it may be a good idea to avoid using any RNN type model for those circuits.

Another major contribution of this work is the definition and development of stability conditions for RNN-based circuit modeling. The usual stability condition defined for general nonlinear systems may only be applied to circuits without feedback, which excludes all circuits that contain ports with mutually dependent voltage and current waveforms. The derivation of a useful stability condition for those excluded circuits is difficult. Nevertheless, a stability condition is derived and applied to a 1-port circuit subjected to Thevenin source, modeled with the ordinary RNN. The stability conditions derived in this work can be easily incorporated with the training process as a form of regularization.

Finally, it is demonstrated that it is best to choose the training data generation method using domain knowledge about the circuit being modeled. A random PWL method is developed as a general training data generation method and it appears to work in many cases. For circuits with special input constraints, the training data generation method should be adjusted to avoid introducing impractical waveforms into the dataset. Also, for circuits with many ports, the random PWL method may run into the "curse of dimensionality" problem since it is a full-factorial method whose required amount of data depends exponentially on the number of ports. To generate training data for such circuits, certain patterns in the input stimuli for the circuit must be exploited for a reduction in the quantity of training data needed.

7.2 Future Work

There are many potential future research projects that can enhance the RNN modeling methodology. A selected list is given in this section.

A potential method of alleviating the vanishing gradient problem would be to train with variable time step data. In principle, the total number of time steps can be reduced by using a small time step for fast-varying portions of the waveforms, and large time step for relatively stable portions. This method may be the most prominent solution to the modeling of circuits with long pure delays, such as transmission lines and certain synchronous sequential logic circuits.

Another important future research topic would be the further development of stability conditions. For multi-port circuits modeled with ordinary RNN or circuits modeled with the Hopfield network, stability conditions for the feedback system may be developed, in principle, with the Lyapunov method. However, it remains unclear if any of such conditions may be easily enforced and incorporated into the training process. On the other hand, no stability conditions are constructed for the GRU. Due to the algebraic complexity of the GRU structure, it already seems unwise to attempt to develop any stability condition, whether any form of feedback is considered. Furthermore, even if any stability conditions can be formulated for GRU, they are nevertheless likely to be too complicated to enforce. If a modeling task demands the use of GRU and the stability of the model is critical, the best method would be to train multiple models and evaluate their stability, then discard the unstable ones. Finally, passivity is a circuit property related to stability, and it may also be worthwhile to attempt to develop passivity conditions for RNN model structures.

APPENDIX A

MATHEMATICAL DERIVATION OF STABILITY CONDITIONS

A sufficient condition for the stability criterion can be derived using the Lyapunov method [42]. The derivation is given in [22] for RNN structure not investigated in this work. In this section, the derivation is performed for the differential representation of the ZIZO ordinary RNN

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) = h^{-1} \cdot [-\boldsymbol{x} + \tanh(W_r \boldsymbol{x} + W_u \boldsymbol{u} + \boldsymbol{b}_u) - \tanh(\boldsymbol{b}_u)] \tag{A.1}$$

In the derivation, notation like $\boldsymbol{a} < \boldsymbol{b}$ means that the inequality holds element-wise, i.e., $\forall i$, $a[i] < b[i]$.

The equilibrium points $(\boldsymbol{x}_s, \boldsymbol{u}_s)$ of the nonlinear system (A.1) are the solutions to the equation $f(\boldsymbol{x}_s, \boldsymbol{u}_s) = \boldsymbol{0}$. For a constant input $\boldsymbol{u}_s$, one obtains the following relationship between $\boldsymbol{x}_s$ and $\boldsymbol{u}_s$:

$$\boldsymbol{x}_s = \tanh(W_r \boldsymbol{x}_s + W_u \boldsymbol{u}_s + \boldsymbol{b}_u) - \tanh(\boldsymbol{b}_u) \tag{A.2}$$

Defining $\boldsymbol{\Gamma} = W_u \boldsymbol{u}_s + \boldsymbol{b}_u$ and making the change of variable $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{x}_s$, one obtains

$$\dot{\boldsymbol{z}} = \frac{1}{h} [-\boldsymbol{z} - \boldsymbol{x}_s + \tanh(W_r \boldsymbol{z} + W_r \boldsymbol{x}_s + \boldsymbol{\Gamma}) - \tanh \boldsymbol{b}_u] \tag{A.3}$$

Substituting (A.2) into (A.3) gives

$$\dot{\boldsymbol{z}} = \frac{1}{h} [-\boldsymbol{z} - \tanh(W_r \boldsymbol{x}_s + \boldsymbol{\Gamma}) + \tanh(W_r \boldsymbol{z} + W_r \boldsymbol{x}_s + \boldsymbol{\Gamma})] \tag{A.4}$$

Define the vector function

$$\boldsymbol{f}(\boldsymbol{w}) = \tanh(\boldsymbol{w} + W_r\boldsymbol{x}_s + \boldsymbol{\Gamma}) - \tanh(W_r\boldsymbol{x}_s + \boldsymbol{\Gamma}) \tag{A.5}$$

Equation (A.4) can now be written as

$$\dot{\boldsymbol{z}} = \frac{1}{h}[-\boldsymbol{z} + \boldsymbol{f}(W_r\boldsymbol{z})] \tag{A.6}$$

To derive a stability condition, the following Lyapunov function is introduced:

$$V(\boldsymbol{z}) = \mathbf{1} \cdot \int_0^{W_r\boldsymbol{z}} \boldsymbol{f}(\boldsymbol{w})d\boldsymbol{w} \tag{A.7}$$

where $\mathbf{1} = [1\,1\,\ldots\,1]$ is a vector of ones. In order to guarantee global asymptotic stability of system (A.4), $V(\boldsymbol{z})$ needs to satisfy the following conditions:

1. $V(\mathbf{0}) = 0$ and $V(\boldsymbol{z}) > 0$ for $\boldsymbol{z} \neq \mathbf{0}$.
2. $V(\boldsymbol{z}) \to \infty$ when $\|\boldsymbol{z}\| \to \infty$.
3. $\dot{V}(\boldsymbol{z}) = \frac{dV}{d\boldsymbol{z}}\dot{\boldsymbol{z}} < 0$ for $\boldsymbol{z} \neq \mathbf{0}$.

Since $\tanh(x)$ is a monotonically increasing function of x, $\boldsymbol{f}(\boldsymbol{w})$ and $\boldsymbol{w}$ must have the same sign component-wise. Therefore,

$$\int_0^{W_r\boldsymbol{z}} \boldsymbol{f}(\boldsymbol{w})d\boldsymbol{w} > 0$$

holds for all $\boldsymbol{z} \neq 0$, and condition 1 is satisfied. Also, from (A.5), it is quite obvious that condition 2 is satisfied. To evaluate condition 3, calculate the time derivative of $V(\boldsymbol{z})$:

$$\dot{V}(\boldsymbol{z}) = \frac{dV}{d\boldsymbol{z}}\dot{\boldsymbol{z}} \tag{A.8a}$$

$$= \boldsymbol{f}^T(W_r\boldsymbol{z})W_r\frac{1}{h}[-\boldsymbol{z} + \boldsymbol{f}(W_r\boldsymbol{z})] \tag{A.8b}$$

$$= \frac{1}{h}[\boldsymbol{f}^T(W_r\boldsymbol{z})W_r\boldsymbol{f}(W_r\boldsymbol{z}) - \boldsymbol{f}^T(W_r\boldsymbol{z})W_r\boldsymbol{z}] \tag{A.8c}$$

Since $0 < \frac{d}{dx}\tanh(x) < 1$ holds for all x, it can be proved that $|\boldsymbol{f}(W_r\boldsymbol{z})| < |W_r\boldsymbol{z}|$ holds for all $\boldsymbol{z} \neq \boldsymbol{0}$, thus

$$\boldsymbol{f}^T(W_r\boldsymbol{z})W_r\boldsymbol{z} > \boldsymbol{f}^T(W_r\boldsymbol{z})\boldsymbol{f}(W_r\boldsymbol{z}) \tag{A.9}$$

Substituting (A.9) into (A.8c) gives

$$\dot{V}(\boldsymbol{z}) < \frac{1}{h}[\boldsymbol{f}^T(W_r\boldsymbol{z})W_r\boldsymbol{f}(W_r\boldsymbol{z}) - \boldsymbol{f}^T(W_r\boldsymbol{z})\boldsymbol{f}(W_r\boldsymbol{z})] \tag{A.10}$$

Thus

$$\dot{V}(\boldsymbol{z}) < \frac{1}{h}\boldsymbol{f}^T(W_r\boldsymbol{z})(W_r - I)\boldsymbol{f}(W_r\boldsymbol{z}) \tag{A.11}$$

For both sides of (A.11), taking the transpose and then averaging with the original equation gives:

$$\dot{V}(\boldsymbol{z}) < \frac{1}{h}\boldsymbol{f}^T(W_r\boldsymbol{z})\left(\frac{W_r + W_r^T}{2} - I\right)\boldsymbol{f}(W_r\boldsymbol{z}) \tag{A.12}$$

From (A.12) it is evident that, to satisfy condition 3, that the symmetric matrix

$$T_0 = \frac{W_r + W_r^T}{2} - I \tag{A.13}$$

must be negative definite, i.e., all eigenvalues of T_0 must be negative. This is a sufficient condition of ABST for the nonlinear system (A.1).

In the derivation, the activation function is assumed to be the hyperbolic tangent. However, the derivation remains valid for any activation function whose derivative is uniformly bounded between 0 and 1. Many commonly used activation functions in neural networks satisfy this condition, including all activation functions listed in Table 3.1.

APPENDIX B

STABILITY ANALYSIS OF THE NONLINEAR RLC CIRCUIT

Consider the nonlinear RLC test circuit connected to a Thevenin source with voltage V_s and resistance R_s. From KCL and KVL, differential equations describing the circuit can be written as

$$i_L = f(v_C) + C\dot{v}_C \tag{B.1a}$$

$$V_s = R_s i_L + L\dot{i}_L + v_C \tag{B.1b}$$

The nonlinear system (B.1) can be written in the following nonlinear state-space form:

$$\dot{v}_C = C^{-1}(-f(v_C) + i_L) \tag{B.2a}$$

$$\dot{i}_L = L^{-1}(-v_C - R_s i_L + V_s) \tag{B.2b}$$

with the state variables being v_C and i_L. For a known V-R combination, the equilibrium of the system can be solved by setting all time-derivatives to zero. Thus, the equilibrium states of the system satisfies

$$v_e + R_s f(v_e) = V_s \tag{B.3a}$$

$$i_e = f(v_e) \tag{B.3b}$$

where v_e is the equilibrium state of v_C and i_e is the equilibrium state of i_L.

To demonstrate the ABST of (B.2), a Lyapunov function $\Phi(v_C, i_L)$ needs to be found that satisfies all the following conditions:

1. $\Phi \geq 0$ for all $v_C, i_L \in \mathbb{R}$ with $\Phi = 0$ if and only if $v_C = v_e$ and $i_L = i_e$.

2. $\Phi \to \infty$ when $v_C^2 + i_L^2 \to \infty$.

3. $\dot{\Phi} = \frac{\partial \Phi}{\partial v_C}\dot{v}_C + \frac{\partial \Phi}{\partial i_L}\dot{i}_L \leq 0$ for all $v_C, i_L \in \mathbb{R}$ with $\dot{\Phi} = 0$ if and only if $v_C = v_e$ and $i_L = i_e$.

In general, it is most natural to attempt to formulate the Lyapunov function as the energy function of the system. Choose

$$\Phi(v_C, i_L) = \frac{C}{2}(v_C - v_e)^2 + \frac{L}{2}(i_L - i_e)^2 \tag{B.4}$$

Obviously, the first two conditions for the Lyapunov function are satisfied by (B.4). Condition 3 can be tested:

$$\begin{aligned}
\dot{\Phi} &= \frac{\partial \Phi}{\partial v_C}\dot{v}_C + \frac{\partial \Phi}{\partial i_L}\dot{i}_L \\
&= C(v_C - v_e)C^{-1}(-f(v_C) + i_L) + L(i_L - i_e)L^{-1}(-v_C - R_s i_L + V_s) \\
&= (v_C - v_e)(-f(v_C) + i_L) + (i_L - f(v_e))(-v_C - R_s i_L + v_e + R_s f(v_e)) \\
&= -(v_C - v_e)(f(v_C) - f(v_e)) - R_s(i_L - i_e)^2
\end{aligned}$$

Obviously, condition 3 for the Lyapunov function is satisfied if $(v_C - v_e)(f(v_C) - f(v_e)) > 0$ for $v_C \neq v_e$. This is equivalent to saying that the $I = f(V)$ is a monotonically increasing function of V. For the nonlinear resistor used in the test case, its I-V relationship $I = 4\tanh(V) + \frac{1}{4}V$ is obviously monotonically increasing. Therefore, the Lyapunov satisfies all three conditions, and consequently the system (B.2) is ABST.

REFERENCES

[1] L. Jin, P. N. Nikiforuk, and M. M. Gupta, “Approximation of discrete-time state-space trajectories using dynamic recurrent neural networks,” *IEEE Transactions on Automatic Control*, vol. 40, no. 7, pp. 1266–1270, July 1995.

[2] A. P. Trischler and G. M. D’Eleuterio, “Synthesis of recurrent neural networks for dynamical system simulation,” *Neural Networks*, vol. 80, pp. 67 – 78, 2016.

[3] *Electromagnetic compatibility (EMC) - Part 4-2: Testing and measurement techniques - Electrostatic discharge immunity test*, IEC 61 000-4-2:2008, 2008.

[4] C. Reiman, N. Thomson, Y. Xiu, R. Mertens, and E. Rosenbaum, “Practical methodology for the extraction of SEED models,” in *EOS/ESD Symposium Proceedings*, Sep. 2015, pp. 1–10.

[5] V. Volterra, *Theory of Functionals and of Integro-differential Equations.* NY: Dover, 1959.

[6] S. A. Maas and A. Crosmun, “Modeling the gate I/V characteristic of a GaAs MESFET for Volterra-series analysis,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 37, no. 7, pp. 1134–1136, July 1989.

[7] F. Filicori and G. Vannini, “Mathematical approach to large-signal modelling of electron devices,” *Electronics Letters*, vol. 27, no. 4, pp. 357–359, Feb 1991.

[8] A. J. M. Kaizer, “Modeling of the nonlinear response of an electrodynamic loudspeaker by a Volterra series expansion,” *Journal of the Audio Engineering Society*, vol. 35, no. 6, pp. 421–433, 1987.

[9] E. Ngoya, N. L. Gallou, J. M. Nebus, H. Buret, and P. Reig, “Accurate RF and microwave system level modeling of wideband nonlinear circuits,” in *IEEE MTT-S International Microwave Symposium Digest*, vol. 1, June 2000, pp. 79–82.

[10] D. Mirri, G. Luculano, F. Filicori, G. Pasini, G. Vannini, and G. P. Gabriella, “A modified Volterra series approach for nonlinear dynamic systems modeling,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 8, pp. 1118–1128, Aug 2002.

[11] A. Zhu, M. Wren, and T. J. Brazil, “An efficient Volterra-based behavioral model for wideband RF power amplifiers,” in *IEEE MTT-S International Microwave Symposium Digest*, vol. 2, June 2003, pp. 787–790.

[12] A. Zhu and T. J. Brazil, “Behavioral modeling of RF power amplifiers based on pruned volterra series,” *IEEE Microwave and Wireless Components Letters*, vol. 14, no. 12, pp. 563–565, Dec 2004.

[13] A. Zhu and T. J. Brazil, “RF power amplifier behavioral modeling using Volterra expansion with Laguerre functions,” in *IEEE MTT-S International Microwave Symposium Digest*, 2005.

[14] A. Zhu, J. C. Pedro, and T. J. Brazil, “Dynamic deviation reduction-based Volterra behavioral modeling of RF power amplifiers,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 54, no. 12, pp. 4323–4332, Dec 2006.

[15] A. A. M. Saleh, “Frequency-independent and frequency-dependent nonlinear models of TWT amplifiers,” *IEEE Transactions on Communications*, vol. 29, no. 11, pp. 1715–1720, November 1981.

[16] M. Abuelma’atti, “Frequency-dependent nonlinear quadrature model for TWT amplifiers,” *IEEE Transactions on Communications*, vol. 32, no. 8, pp. 982–986, August 1984.

[17] J.-T. Hsu and K. D. T. Ngo, “Behavioral modeling of the IGBT using the Hammerstein configuration,” *IEEE Transactions on Power Electronics*, vol. 11, no. 6, pp. 746–754, Nov 1996.

[18] F. Alonge, F. D’Ippolito, F. M. Raimondi, and S. Tumminaro, “Nonlinear modeling of DC/DC converters using the Hammerstein’s approach,” *IEEE Transactions on Power Electronics*, vol. 22, no. 4, pp. 1210–1221, July 2007.

[19] J. Moon and B. Kim, “Enhanced Hammerstein behavioral model for broadband wireless transmitters,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 59, no. 4, pp. 924–933, April 2011.

[20] I. S. Stievano, I. A. Maio, and F. G. Canavero, “Parametric macromodels of digital I/O ports,” *IEEE Transactions on Advanced Packaging*, vol. 25, no. 2, pp. 255–264, May 2002.

[21] D. Luongvinh and Y. Kwon, “Behavioral modeling of power amplifiers using fully recurrent neural networks,” in *IEEE MTT-S International Microwave Symposium Digest*, June 2005.

[22] Y. Cao, R. Ding, and Q.-J. Zhang, "State-space dynamic neural network technique for high-speed IC applications: modeling and stability analysis," *IEEE Transactions on Microwave Theory and Techniques*, vol. 54, no. 6, pp. 2398–2409, June 2006.

[23] Y. Cao and Q. J. Zhang, "A new training approach for robust recurrent neural-network modeling of nonlinear circuits," *IEEE Transactions on Microwave Theory and Techniques*, vol. 57, no. 6, pp. 1539–1553, June 2009.

[24] Y. Cao, X. Chen, and G. Wang, "Dynamic behavioral modeling of nonlinear microwave devices using real-time recurrent neural network," *IEEE Transactions on Electron Devices*, vol. 56, no. 5, pp. 1020–1026, May 2009.

[25] M. Rawat, K. Rawat, and F. M. Ghannouchi, "Adaptive digital predistortion of wireless power amplifiers/transmitters using dynamic real-valued focused time-delay line neural networks," *IEEE Transactions on Microwave Theory and Techniques*, vol. 58, no. 1, pp. 95–104, Jan 2010.

[26] V. Devabhaktuni, C. F. Bunting, D. Green, D. Kvale, L. Mareddy, and V. Rajamani, "A new ANN-based modeling approach for rapid EMI/EMC analysis of PCB and shielding enclosures," *IEEE Transactions on Electromagnetic Compatibility*, vol. 55, no. 2, pp. 385–394, April 2013.

[27] B. Mutnury, M. Swaminthan, M. Cases, N. Pham, D. N. de Araujo, and E. Matoglu, "Macromodeling of nonlinear transistor-level receiver circuits," *IEEE Transactions on Advanced Packaging*, vol. 29, no. 1, pp. 55–66, Feb 2006.

[28] S. A. Sadrossadat, P. Gunupudi, and Q. Zhang, "Nonlinear electronic/photonic component modeling using adjoint state-space dynamic neural network technique," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 5, no. 11, pp. 1679–1693, Nov 2015.

[29] G. Chrisikos, C. J. Clark, A. A. Moulthrop, M. S. Muha, and C. P. Silva, "A nonlinear ARMA model for simulating power amplifiers," in *IEEE MTT-S International Microwave Symposium Digest*, vol. 2, June 1998, pp. 733–736.

[30] D. Drmanac, B. Bolin, and L. Wang, "A non-parametric approach to behavioral device modeling," in *2010 11th International Symposium on Quality Electronic Design (ISQED)*, March 2010, pp. 284–290.

[31] T. Liu, S. Boumaiza, and F. M. Ghannouchi, "Dynamic behavioral modeling of 3G power amplifiers using real-valued time-delay neural networks," *IEEE Transactions on Microwave Theory and Techniques*, vol. 52, no. 3, pp. 1025–1033, Mar. 2004.

[32] V. Rizzoli, A. Neri, D. Masotti, and A. Lipparini, "A new family of neural network-based bidirectional and dispersive behavioral models for nonlinear RF/microwave subsystems," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 12, no. 1, pp. 51–70, 2002.

[33] M. Isaksson, D. Wisell, and D. Ronnow, "Wide-band dynamic modeling of power amplifiers using radial-basis function neural networks," *IEEE Transactions on Microwave Theory and Techniques*, vol. 53, no. 11, pp. 3422–3428, Nov 2005.

[34] M. Kraemer, D. Dragomirescu, and R. Plana, "Nonlinear behavioral modeling of oscillators in VHDL-AMS using artificial neural networks," in *2008 IEEE Radio Frequency Integrated Circuits Symposium*, June 2008, pp. 689–692.

[35] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, Nov. 2016.

[36] A. Graves, M. Liwicki, S. Fernndez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, May 2009.

[37] E. Grund and R. Gauthier, "VF-TLP systems using TDT and TDRT for kelvin wafer measurements and package level testing," in *EOS/ESD Symposium Proceedings*, 2004.

[38] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *ArXiv e-prints*, Dec. 2012.

[39] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[40] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[41] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *ArXiv e-prints*, June 2014.

[42] H. K. Khalil, *Nonlinear Systems.* New York, NY: Macmillan, 1992.

[43] V. Sundarapandian, "Global asymptotic stability of nonlinear cascade systems," *Applied Mathematics Letters*, vol. 15, no. 3, pp. 275–277, April 2002.

[44] T. Chu and C. Zhang, "New necessary and sufficient conditions for absolute stability of neural networks," in *2005 International Conference on Control and Automation*, vol. 1, June 2005, pp. 593–598.

[45] E. Kaszkurewicz and A. Bhaya, "On a class of globally stable neural circuits," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 41, no. 2, pp. 171–174, Feb 1994.

[46] M. Forti and A. Tesi, "New conditions for global stability of neural networks with application to linear and quadratic programming problems," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 7, pp. 354–366, July 1995.

[47] R. Merrill and E. Issaq, "ESD design methodology," in *EOS/ESD Symposium Proceedings*, 1993.