

American University in Cairo

AUC Knowledge Fountain

Theses and Dissertations

Student Research

Spring 6-21-2022

Machine Learning Applications to Static Timing Analysis

Waseem Mohamed Raslan

Follow this and additional works at: <https://fount.aucegypt.edu/etds>



Part of the [Electrical and Electronics Commons](#), [Electronic Devices and Semiconductor Manufacturing Commons](#), and the [VLSI and Circuits, Embedded and Hardware Systems Commons](#)

Recommended Citation

APA Citation

Raslan, W. (2022). *Machine Learning Applications to Static Timing Analysis* [Doctoral Dissertation, the American University in Cairo]. AUC Knowledge Fountain.

<https://fount.aucegypt.edu/etds/1920>

MLA Citation

Raslan, Waseem Mohamed. *Machine Learning Applications to Static Timing Analysis*. 2022. American University in Cairo, Doctoral Dissertation. *AUC Knowledge Fountain*.

<https://fount.aucegypt.edu/etds/1920>

This Doctoral Dissertation is brought to you for free and open access by the Student Research at AUC Knowledge Fountain. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AUC Knowledge Fountain. For more information, please contact thesisadmin@aucegypt.edu.



*Machine Learning Applications to Static Timing
Analysis*

A THESIS SUBMITTED BY

Waseem Mohamed Raslan

TO THE

*Department of Electronics and Communications
Engineering*

May 2022

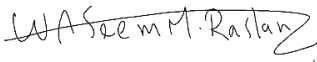
*in partial fulfillment of the requirements for the degree of
Doctor of Philosophy of Science in Electronics and Communications
Engineering*

Declaration of Authorship

I, Waseem Mohamed Raslan, declare that this thesis titled, “[Thesis title]” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:



Waseem Mohamed Raslan

Date:

May 2022

Abstract

Neural networks had been subject of continuous research since the sixties of last century. It went through ups and downs in research and industry potential until the last decade. Three main factors contributed to the rise of deep learning and artificial intelligence as instrumental tools in future; availability of enormous data, availability of computational power needed to train big data; and devising training algorithms that can train deep neural networks. Using feed-forward neural networks proved to be successful in regression and classification analysis. Modeling complex cell behavior is critical for accurate static timing analysis. Accounting for larger wire delays, noise and inductance effects on smaller transistor feature sizes resulted in the need to capture more complex waveforms during library cell characterization process. The more complex waveforms, the more waveform samples that need to be stored on disk to capture waveform overshoots, undershoots and multiple crossings. Increased technology file size can have drastic effects on the performance of digital design flow processes and static timing analysis that depend on these library files. Effective current source model, ECSM, and composite current source, CCS, waveform data compression became a necessity to reduce the size of technology files and increase the accuracy of the cell characterization data. Huge waveform data needed for current source models explodes technology file size and degrades design flow performance. We used deep learning nonlinear Autoencoders to compress voltage and current waveforms and compared them with singular component analysis approach. Autoencoders gave $\sim 1.67x$ compression ratio for voltage waveforms better than, singular value decomposition, SVD, approach and between $\sim 45x$ and $\sim 55x$ better compression ratio

that lossless compression techniques like gzip and bz2. Autoencoders achieved $\sim 1.7\times$ compression ratio for complex rising-edge current waveforms at model loss of $7.6e-5$ and comparable results to SVD approach for the falling-edge waveforms. However, SVD remains more computationally efficient than Autoencoders. Deep learning non-linear delay model, DL-NLDM, is proposed to replace the standard 7×7 non-linear delay modeling lookup tables, NLDM-LUT, that hold the delay information according to the input transition and effective capacitance values. The proposed DL-NLDM performed better than the standard 7×7 NLDM-LUT tables in average, standard deviation and maximum percentage errors compared to SPICE simulation. Building on both DL-NLDM and waveform compression, a combined DL-NLDM ECSM waveform model is proposed to produce both delay/transition time information as well as the compressed waveform parameters. Experiments show that separate DL-NLDM and encoded ECSM waveform parameters are better than the combined ones. In addition, deep learning waveform delay model, DL-WFDM, is proposed to radically change transition/delay propagation to a full waveform propagation that can be used to measure the delay or perform ECSM delay calculations. Experiments also show that separate DL-NLDM and encoded ECSM waveform parameters still perform better, in delay errors, than the proposed DL-WFDM models.

Another type of neural network is used in this research to perform model order reduction, MOR, of linear time invariant, LTI, systems. Recurrent neural networks, RNN, where outputs of neurons are fed back into neurons inputs proved to help in modeling time series data in which future output is dependent on experience. Obtaining accurate and less computational demanding reduced models is a continuous challenge with complex

systems. We propose a structured recurrent neural network, S-RNN, that can model LTI single input single output, SISO, systems of any order. The weights of the trained S-RNN are directly mapped to the discrete state space equations of the system. We showed how to obtain the continuous time transfer function of the reduced system from the trained S-RNN weights. Using this S-RNN model outperformed other model order reduction techniques reported in selected literature. The proposed S-RNN reduced a complex system of 598 states is reduced to a 10th order system at $9.04e-6$ mean-square-error. SISO 4th order outperformed reported results of other MOR techniques. The S-RNN is extended to model single input single output, SIMO, LTI of any number of output and any system order. Using this RNN SIMO network, RLC interconnect of 108 states was reduced to a 5th system at $9.1e-4$ mean square error. S-RNN is also shown to be able to model multiple input multiple output, MIMO, systems with better results than modeling individual MIMO input output relationship.

Acknowledgements

First, I would like to thank Allah for helping me through finishing this thesis work.

I would like also to thank my wonderful wife Marwa, my cool son Ahmed, and my lovely daughter Farida, for their endless support and encouragement through the long years I spent pursuing my PhD degree not to mention the other years I spent studying for my MSc and MBA degrees. I hope I had set a good example for Ahmed and Farida showing that pursuing knowledge and contributing to research has no age limitation.

I would like to thank Prof. Dr. Ashraf Salem for his continuous support, encouragement and literally pushing me to complete this research work.

In addition, I really appreciate and acknowledge the time and effort my internal and external examiners have offered to assess this research on such a short notice.

Finally, my sincere appreciation for Prof. Dr. Yehea Ismail, my dear supervisor for his tremendous support and confidence in addition to his valuable guidance throughout the research course. I would like to thank Dr. Yehea for being very patient with my extended research period due to work and personal commitments.

Contents

Declaration of Authorship	1
Abstract	2
Acknowledgements	5
Contents	6
List of Figures and Tables	9
List of Abbreviations	14
Chapter 1	16
Introduction	16
1.1 Motivation.....	16
1.2 Thesis Statement.....	17
1.3 Thesis Contribution	17
1.4 Thesis Organization.....	19
Chapter 2	22
Machine Learning Applications in Electronic Design Automation.....	22
2.1 Design Space Exploration.....	23
2.2 Power Estimation.....	24
2.3 Functional Verification.....	25
2.4 Timing Analysis	26
2.5 Physical Design	27
2.6 Cell Characterization.....	28
2.7 Routing	28
2.8 Other EDA Applications.....	29
Chapter 3	30
Cell Characterization Models.....	30
3.1 Overview.....	30
3.2 Liberty File Format	31
3.3 Liberty File Structure.....	31
3.4 Non-linear Delay Model, NLDM.....	32
3.4 Current Source Model Waveforms Background	34
3.5 Current Source Cell Receiver Model.....	35

Chapter 4	37
Deep Learning Autoencoder-based Compression for Current Source Model Waveforms ..	37
4.1 Overview	37
4.2 Autoencoder-based Waveform Data Compression	39
4.3 Waveform Data Compression Results	43
4.4 Conclusions.....	49
Chapter 5	51
Deep Learning Cell Driver Delay Modeling	51
5.1 Overview	51
5.2 DL-NLDM: Deep Learning NLDM Cell Delay Model	53
5.3 Combined DL-NLDM ECSM Waveform Cell Delay Model.....	55
5.4 DL-WFDM: Deep Learning Waveform-Delay Model	58
5.5 Conclusions.....	62
Chapter 6	64
S-RNN MOR: Structured Recurrent Neural Network Model Order Reduction for SISO, SIMO and MIMO LTI Systems.....	64
6.1 Overview	64
6.2 Extracting Continuous-time Transfer Functions of RNN MOR Models.....	65
6.3 Model Order Reduction of SISO LTI Systems	66
6.4 SISO LTI RNN Modeling Experiments.....	69
6.5 Model Order Reduction of SIMO LTI Systems.....	74
6.6 Experiment: Modeling RLC Interconnect SIMO LTI RNN	75
6.7 Model Order Reduction of MIMO LTI Systems	79
6.8 S-RNN MIMO Model Order Reduction Experiments	84
6.9 S-RNN Model Order Reduction Applications.....	94
Chapter 7	97
Conclusion and Future Work.....	97
References	100
APPENDIX A	106
Artificial Intelligence, Machine Learning and Deep Learning Background.....	106
A.1 Rule-based Systems	107
A.2 Machine Learning, ML.....	108
A.3 Artificial Neural Networks, ANN.....	119
A.4 Machine and Deep Learning Applications	126
APPENDIX B	127
Model Order Reduction of Non-Linear Systems using Recurrent Neural Networks.....	127
B.1 Overview	127

B.2 One-dimensional non-linear Schrodinger equation	128
B.3 POD-based reduced order model	129
B.4 RNN Model Order Reduction Results.....	129
APPENDIX C	142
Detailed trained models result.....	142
C.1 Autoencoding fixed time 1000-points ECSM Waveforms Detailed Results.....	142
C.2 Autoencoding variable time 50,100,150-points ECSM Waveforms Detailed Results	143
C.3 Autoencoding 21-points ECSM Voltage Waveforms.....	149
C.4 NLDM-LUT Results.....	151
C.5 DL-WFDM results.....	151
APPENDIX D.....	153
Published and accepted papers.....	153

List of Figures and Tables

FIGURE 2-1 Using BOA to optimize design parameters [Park'16]	23
FIGURE 2-2 Small-World NoC energy-efficient and reliable 3D optimization [Das'17]	24
FIGURE 2-3 Power estimation using CLSLR regression technique [Zheng'15]	25
FIGURE 2-4 Functional verification to filter and optimize test cases [Wang'15]	26
FIGURE 2-5 Grouping of failing coverage groups having common root causes using K-Means clustering [Mandouh'16]	26
FIGURE 2-6 SI for free proposed model [Kahng'15]	27
FIGURE 2-7 Hotspot detection using CNN [Yang'17]	28
FIGURE 2-8 ML-based routing model using Gaussian process regression [Chang'17]	29
FIGURE 3-1 Cell Characterization Setup	32
FIGURE 3-2 NLDM Rise/Fall delay time LUT example	33
FIGURE 3-3 NLDM Transition LUT example	33
FIGURE 3-4 CCS rising-edge current waveform example	35
FIGURE 3-5 CC falling-edge current waveform example	35
FIGURE 3-6 CSM for simple inverter	36
FIGURE 3-7 CSM of a NOR2 Cell	36
FIGURE 4.1 Cell Characterization Transistor Level Simulation Setup	40
FIGURE 4.2 1000-points uniform time sampled waveform	40
FIGURE 4-3 Varying Time, 50, 100, 150 Sampling Points	41
Table 4.1 Autoencoder Models for Parameter P = 16,8,4,2,1	42
Table 4.2 Best Autoencoding 1000-points ECSM Falling-Edge Waveform Results Against bz2, gzip and SVD techniques	44
Table 4.3 Best Autoencoding 1000-points ECSM Rising-Edge Waveform Results Against bz2, gzip and SVD techniques	44
FIGURE 4-4 Falling and rising decoded waveforms against Spice waveforms	45
Table 4.4 Best Autoencoding 150-points ECSM Waveform Results Against bz2, gzip and SVD techniques	46
Table 4.5 Best Autoencoding 150-points ECSM Waveform Results Against bz2, gzip and SVD techniques	46
Table 4.6 Autoencoding 50,100,150,1000-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters	46
Table 4.7 Autoencoding 50,100,150,1000-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters	47
Table 4.8 Autoencoding 1000-points Falling-Edge CCS Waveform Results at Different Number of Encoding Parameters	47
Table 4.9 SVD 1000-points Falling-Edge CCS Waveform Results at Different Sigma Rank Number	48
FIGURE 4-5 DECODED 8-P CCS FALLING-EDGE NORMALIZED AND SCALED CURRENT WAVEFORM VS. SPICE WAVEFORM	48
Table 4.10 Autoencoding 1000-points Rising-Edge CCS Waveform Results at Different Number of Encoding Parameters	48

Table 4.11 SVD 1000-points Rising-Edge CCS Waveform Results at Different Sigma Rank Number.....	49
FIGURE 4-6 DECODED 16-P CCS RISING-EDGE NORMALIZED AND SCALED CURRENT WAVEFORM VS. SPICE WAVEFORM.....	49
Table 5.1 Deep Learning Model Structure	53
Table 5.2 DL-NLDM best trained NOR models Mean/Max Percentage Error rates versus different LUT sizes	54
Table 5.3 DL-NLDM best trained INV models Mean/Max Percentage Error rates versus different LUT sizes	54
Table 5.4 DL-NLDM best trained NAND models Mean/Max Percentage Error rates versus different LUT sizes	55
Table 5.5 Combined DL NLDM-ECSM Falling-edge NOR Model.....	56
Table 5.6 Combined DL NLDM-ECSM Falling-edge INV Model	57
Table 5.7 Combined DL NLDM-ECSM Falling-edge NAND Model.....	57
Table 5.8 Combined DL NLDM-ECSM Rising-edge NOR Model.....	57
Table 5.9 Combined DL NLDM-ECSM Rising-edge INV Model	58
Table 5.10 Combined DL NLDM-ECSM Rising-edge NAND Model.....	58
Table 5.11 DL-WFDM NOR Falling-edge Model.....	60
Table 5.12 DL-WFDM INV Falling-edge Model	60
Table 5.13 DL-WFDM NAND Falling-edge Model	60
Table 5.14 DL-WFDM NOR Rising-edge Model.....	61
Table 5.15 DL-WFDM INV Rising-edge Model	61
Table 5.16 DL-WFDM NAND Rising-edge Model	61
Table 5.17 DL-WFDM Multi-stage analysis results	62
FIGURE 6-1 S-RNN Model Training and Continuous TF Extraction Flow	66
FIGURE 6-2 Second Order Discrete SISO System RNN Implementation.....	67
FIGURE 6-3 Third Order Discrete SISO System RNN Implementation.....	68
FIGURE 6-4 N Order Discrete SISO System RNN Implementation	69
FIGURE 6-5 Step response of G-Orig, 4th Order S-RNN and Other Model Order Reduction Techniques	70
Table 6.1 MSE Comparison of S-RNN Performance with Other MOR Techniques	71
FIGURE 6-6 Step response of original “Eady” system and the 5th order MOR.....	71
FIGURE 6-7 Pole-zero map of original and 5th order MOR	72
FIGURE 6-8 Step response of original Eady system and the 10th order	72
Table 6.2 MSE Comparison of 5th and 10th Order S-RNN Models	73
FIGURE 6-9 Pole-zero plot of original and reduced 10th order system	73
FIGURE 6-10 Step Response of the reduced 3rd order model.....	74
FIGURE 6-11 Pole-zero plot of the 3rd order model	74
FIGURE 6-12 Nth Order, Single Input, O Outputs Discrete MIMO System Implementation.....	75
FIGURE 6-13 RLC Spice sub-circuit.....	76
FIGURE 6-14 RLC Transmission Line, v(3), v(8), v(12), v(14)	76
FIGURE 6-15 v(3), v(8) Spice and 2nd order SIMO MOR response	76
FIGURE 6-16 v(12), v(14) Spice and 2nd order SIMO MOR response	77
FIGURE 6-17 v(3), v(8) spice and 5th order SIMO MOR response.....	77
FIGURE 6-18 v(12), v(14) spice and 5th order SIMO MOR response.....	77
FIGURE 6-19 v(3), v(8) spice and 5th order SISO MOR response	78
FIGURE 6-20 v(12), v(14) spice and 5th order SISO MOR response	78
Table 6.3 MSE Comparison of 3rd and 5th Order SIMO and 5th Order SISO S-RNN Models....	78
FIGURE 6-21 Second Order, 2 Inputs, 2 Outputs Discrete MIMO System RNN Implementation	79

FIGURE 6-22 Third Order, 2 Inputs, 2 Outputs Discrete MIMO System RNN Implementation.....	80
FIGURE 6-23 Nth Order, M Inputs, O Outputs Discrete MIMO System Implementation.....	81
FIGURE 6-24 Step response of the original and fourth order reduced MIMO System	83
FIGURE 6-25 Sequence input to generate training MIMO data	83
FIGURE 6-26 Response of the Original System to be Used in Training	84
FIGURE 6-27 Step response of the original and second order reduced MIMO System	85
FIGURE 6-28 Pole-zero of the original and second order reduced MIMO System.....	85
FIGURE 6-29 Step response of the original and third order reduced MIMO System.....	86
FIGURE 6-30 Pole-zero of the original and third order reduced MIMO System	87
Table 6.4 MSE Comparison of MIMO 2nd and 3rd Order S-RNN.....	87
FIGURE 6-31 Response of the Original System to be Used in Training	88
FIGURE 6-32 Step response of the original and 3rd order reduced MIMO System.....	89
FIGURE 6-33 Step response of the original and 4th order reduced MIMO System	90
FIGURE 6-34 Pole-zero plot of the original and 3rd order reduced MIMO System	90
Table 6.5 MSE Comparison of S-RNN of 3rd and 4th Models Performance	91
FIGURE 6-35 Step response of the original and fourth order input 1 to output 1 reduced MIMO System.....	91
FIGURE 6-36 Step response of the original and fourth order input 1 to output 2 reduced MIMO System.....	92
FIGURE 6-37 Step response of the original and fourth order input 2 to output 1 reduced MIMO System.....	93
FIGURE 6-38 Step response of the original and fourth order input 2 to output 2 reduced MIMO System.....	93
Table 6.6 MSE Comparison of S-RNN of 3rd and 4th Models.....	94
FIGURE 6-39 Using RNN model order reduction in black box identification	95
FIGURE A-1 Artificial Intelligence Categories Relationships [Goodfellow'16].....	106
FIGURE A-2 Artificial Intelligence Categories [Goodfellow'16].....	107
FIGURE A-3 Q-learning Reinforced Learning	108
FIGURE A-4 ML Experience [Wang'17].....	110
FIGURE A-5 Machine Learning Over-fitting and Under-fitting [Goodfellow'16]	112
FIGURE A-6 Machine Learning Capacity versus Training and Generalization Error Rates [Goodfellow'16].....	112
FIGURE A-7 Linear Regressions [Eremenko'17]	114
FIGURE A-8 Start with plain dataset [Eremenko'17].....	114
FIGURE A-9 Categorize data based on information entropy [Eremenko'17]	114
FIGURE A-10 Decision Tree formation [Eremenko'17].....	115
FIGURE A-11 Support Vector Machine in Classification [Eremenko'17]	115
FIGURE A-12 K-nearest neighbor [Eremenko'17].....	116
FIGURE A-13 Naïve Bayes Classification Steps	117
FIGURE A-14 K-Means Clustering Steps [Eremenko'17].....	118
FIGURE A-15 Hierarchical Clustering Steps [Eremenko'17].....	119
FIGURE A-16 Single Perceptron	119
FIGURE A-17 Threshold and Rectifier Functions.....	120
FIGURE A-18 Sigmoid and Hyperbolic Functions.....	120
FIGURE A-19 Artificial Neural Network.....	121
FIGURE A-20 Back Propagation Learning Algorithm.....	121
FIGURE A-21 Gradient descent algorithm [Eremenko'17]	122
FIGURE A-22 Recurrent Neural Network.....	122
FIGURE A-23 Recurrent Neural Network.....	123
FIGURE A-24 Convolutional Neural Networks, CNN [Eremenko'17].....	124
FIGURE A-25 TRAINING of DEEP LEARNING AUTOENCODER.....	125
FIGURE A-26 AUTOENCODER Split to Encoder and Decoder.....	125
FIGURE B-1 Matlab Simulation with Sech(x) initial function	130

FIGURE B-2 SVD Analysis.....	130
FIGURE B-3 First three modes plot.....	131
FIGURE B-4 One Mode POD MOR.....	131
FIGURE B-5 Second order S-RNN MOR.....	132
FIGURE B-6 SVD analysis of Second order S-RNN solution.....	132
FIGURE B-7 Matlab Simulation with $2 \cdot \text{Sech}(x)$ initial function.....	133
FIGURE B-8 SVD analysis of Matlab solution	134
FIGURE C-1 DECODED ECSM FALLING AND RISING-EDGE WAVEFORMS VS. SPICE WAVEFORMS	142
Table C.1 Autoencoding 1000-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters	142
Table C.2 SVD 1000-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number.....	143
Table C.3 Autoencoding 1000-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters	143
Table C.4 SVD 1000-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number.....	143
Table C.5 Autoencoding 50-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters	143
Table C.6 SVD 50-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number.....	144
FIGURE C-2 SPICE vs reconstructed 50-points sampled falling edge waveform	144
Table C.7 Autoencoding 50-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters	144
Table C.8 SVD 50-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number.....	145
FIGURE C-3 SPICE vs reconstructed 50-points sampled rising edge waveform	145
Table C.9 Autoencoding 100-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters	145
Table C.10 SVD 100-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number.....	146
FIGURE C-4 SPICE vs reconstructed 100-points sampled falling edge waveform	146
Table C.11 Autoencoding 100-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters	146
Table C.12 SVD 100-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number.....	147
FIGURE C-5 SPICE vs reconstructed 100-points sampled rising edge waveform	147
Table C.13 Autoencoding 150-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters	147
Table C.14 SVD 150-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number.....	148
FIGURE C-6 SPICE vs reconstructed 150-points sampled falling edge waveform	148
Table C.15 Autoencoding 150-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters	148
Table C.16 SVD 150-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number.....	148
FIGURE C-7 SPICE vs reconstructed 150-points sampled rising edge waveform	149
Table C.17 Autoencoding 21-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters	150

Table C.18 SVD 21-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number.....	150
Table C.19 Autoencoding 21-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters	150
Table C.20 SVD 21-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number.....	150
Table C.21 Falling Edge Mean Percentage Error NLDM-LUT versus Spice.....	151
Table C.22 Rising Edge Mean Percentage Error NLDM-LUT versus Spice.....	151
Table C.23 DL-WFDM Model Training Results using 1000-points Sampled Waveforms	151
Table C.24 DL-WFDM Model Training Results using 150-points Sampled Waveforms.....	151

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BOA	Bayesian Optimization Algorithm
BP	Back Propagation Algorithm
BT	Balanced Truncation
CCS	Composite Current Source
CNN	Convolutional Neural Network
CSM	Current Source Model
DL	Deep Learning
DL-NLDM	Deep Learning Nonlinear Delay Model
DL-WFDM	Deep Learning Waveform Delay Model
ECSM	Effective Current Source Model
EDA	Electronic Design Automation
IV	Instrumental Variable
J-RNN	Jordan-Recurrent Neural Network
LTI	Linear Time-Invariant
MOR	Model Order Reduction
ML	Machine Learning
MLP	Multi-Layer Perceptron
MIMO	Multiple-input multiple-output
NLDM	Nonlinear Delay Model
PCA	Principal Component Analysis

PDN	Power Distribution Network
RNN	Recurrent Neural Network
S-RNN	Structured Recurrent Neural Network
SI	Signal Integrity
SISO	Single-input single-output
SIMO	Single-input multiple-output
STA	Static Timing Analysis
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TSV	Through Silicon Via

Chapter 1

Introduction

1.1 Motivation

Electronic design automation, EDA, tools are challenged with complex designs and enormous data that are getting generated during the design flow starting from requirements down to physical implementation. Machine and deep learning techniques proved to be very useful handling extremely large data during the last decade. Advent of new training algorithms, existence of very powerful processing power and very large data availability allowed deep learning to flourish and be used in practical applications in different domains. Data explosion forced big companies like Google and Microsoft to use AI/ML techniques in their search engines, image recognition, and movie analysis applications. The momentum behind autonomous cars, robotics and drones forced companies like Google and Tesla to use AI/ML techniques to empower computer vision and decision-making algorithms. IoT, Cloud Computing, 5G and Industry 4.0 are all driving big companies like Amazon, Microsoft, Siemens, Oracle, and IBM to allocate large budgets to AI/ML research to be able to handle the next wave of data explosion expected with all these technologies.

Machine and deep learning techniques penetrated design flows and EDA tools. Many EDA companies are investing in machine learning, ML, applications anticipating that only ML-enabled EDA applications will be able to handle the complexity of the next generation semiconductor

design process. This research focuses on ML and DL applications in EDA applications to explore what is happening in the industry and academia as well as to contribute to this important field of research.

1.2 Thesis Statement

This thesis starts with surveying machine learning applications to electronic design automation.

This thesis work focused on deep learning applications in cell delay modeling to increase the accuracy of delay models without increasing the size of the cell technology file. In addition, this thesis proposes deep learning applications for model order reduction of linear-time invariant systems to model white box complex systems and black box unknown systems given the original system step response.

1.3 Thesis Contribution

The main contributions of this thesis are:

- Proposed deep learning non-delay model, DL-NLDM, to model cell delays using DL neural networks. This approach reduced the two lookup tables, NLDM-LUT, characterizing cell delay and transition time to one DL model for rising edge and another DL model for falling edge. For falling edge model, the proposed DL-NLDM outperforms 7x7 NLDM-LUT average percentage errors by $\sim 1.24x$ delay time and $\sim 1.55x$ transition time over the selected wide range of input values. The same DL-NLDM model outperforms 100x100 NLDM-LUT maximum percentage errors by $\sim 2.37x$ delay time and by $\sim 2x$ for transition time. For rising edge model, the proposed DL-NLDM outperforms 7x7 NLDM-LUT average percentage errors by $\sim 5x$ for both delay and transition time and outperforms 100x100 NLDM-LUT maximum percentage errors by $\sim 3.19x$ for delay time and by $\sim 2.75x$ for transition time. The proposed rising edge model gives less than 1.41% average delay and 4.86% average transition time percentage error compared to SPICE.

DL-NLDM retrieving time outperformed this of 7x7 NLDM-LUT by 1.69x, and outperformed 100x100 NLDM-LUT by 12x using the outlined Python implementation.

- Proposed DL Autoencoder-based compression technique for ECSM voltage waveforms to increase the waveform fidelity without increasing the technology file size. This DL compression technique outperformed the nearest rank SVD technique compression ratio by $\sim 1.67x$, and by $\sim 1.78x$ for 150-points sampled waveforms. Two params Autoencoders compression ratio outperforms lossless compression techniques by $\sim 45x$ to $55x$ factor. On the other hand, nearest SVD rank outperforms 2 params encoding decompression time by $\sim 12.25x$ and $\sim 17.34x$ for 1000-points and 150-points sampled waveforms respectively. For 1000-points sampled waveforms, 2 params Autoencoders models give between 0.4% to 0.85% standard deviation of percentage errors compared to SPICE simulation. For 150-points sampled waveforms, 2 params Autoencoders models give between 0.2% to 0.4% standard deviation of percentage errors compared to SPICE simulation. Encoding with 4 params Autoencoders models are generally $\sim 2x$ better in error rates compared to the 2 params models on the expense of lower compression ratio.
- Proposed compressing CCS voltage time waveforms using the same DL Autoencoder technique. The proposed Autoencoders give 1.7x better compression ratio results than the nearest rank SVD for 1000-points sampled rising-edge current waveforms. Autoencoders give 26.5x compression ratio compared to 15.4x SVD reported compression ratio. SVD compression gave slightly better results for falling-edge current waveforms, 30.8x versus 29.34x Autoencoder compression ratio.
- Proposed a combined DL-NLDM ECSM waveform compressed model trained to produce two-parameters encoded output waveforms given input transition time and input

capacitive load values. A combined DL-NLDM ECSM model is trained for each edge and for each cell type. Experiment results show that separate DL-NLDM and ECSM waveform parameters perform better than the combined model. However, the combined model remains better than the maximum percentage delay and transition time errors of the standard 7x7 NLDM-LUT over the selected wide cell characterization values range.

- Proposed modeling cell delays using DL waveform delay model, DL-WFDM, trained to produce two-parameters encoded output waveforms given two-parameters encoded input waveforms and capacitive load input values. Experiments show that separate DL-NLDM and ECSM waveform parameters perform better than the proposed DL-WFDM model, however there is a great potential to improve this new modeling methodology performance results.
- Proposed and trained a structured recurrent neural network, S-RNN, that models Nth order LTI SISO systems. Proposed and trained other S-RNN network structures to model Nth order LTI SIMO systems of any number of outputs O. Proposed RNN network structure to model Nth order MIMO LTI systems of any number of input M and output O. Reconstructed the continuous time state-space model from the weights of the trained S-RNN models.

1.4 Thesis Organization

This thesis is organized as it follows: Chapter 2 of this thesis surveys the most notable machine learning research activities in electronic design automation. Chapter 3 provides background information on cell characterization models which most of this research focuses on to apply deep learning applications. Chapter 4 details the proposed DL Autoencoder-based waveform compression technique and the compression results compared with other compression techniques. Chapter 5 presents different proposed deep learning cell delay modeling techniques

and experiment results. Chapter 6 shows another proposed application of deep learning technique - structured recurrent neural networks, S-RNN, to perform model order reduction of complex SISO, SIMO and MIMO linear time-invariant, LTI, systems. Chapter 6 lists several experiment results for different types of complex systems. Finally, chapter 7 contains thesis conclusions and future work followed by the list of references.

Chapter 2

Machine Learning Applications in Electronic Design Automation

There are many driving forces behind the machine learning success in the recent decade, on top to these forces is the huge available data that need deeper analysis to extract useful information. Google and Bing search engines and other big names like Amazon, and Netflix invested heavily in machine and deep learning for data analysis, image recognition, and movie analysis. Hi-tech companies developing autonomous cars and drones invested in image processing and computer vision. Companies driving IoT, cloud computing and the promises of Industry 4.0 like Amazon, Siemens, IBM and Oracle are investing in data analytics.

ML/DL techniques are penetrating EDA applications. Focus on optimization, classification, and image recognition problems with large parameters. Search for optimal design with acceptable accuracy, reliable, cost effective and scalable algorithms. DAC, the most notable IEEE EDA conference focus started to dedicate special track for machine learning application in EDA applications for papers and tutorials. Several joint industry and academia centers are founded to focus on ML applications in EDA like CAEML, "Center for Advanced Electronics Through Machine Learning". EDA companies like Synopsys, Cadence and Siemens EDA are investing in in-house ML enabled applications as well as investing in acquiring high potential startups. Many EDA startups focusing on machine learning had successful launch and some have been acquired by big EDA names. Many recent publications in the last five years across all design life cycle. The following sections explores the recent research activities of machine and deep learning in

electronic design automation domain.

2.1 Design Space Exploration

3D-IC design space application: Heterogeneous die integration provides More than Moore, however thermal gradient and TSV reliability limit 3D-IC benefits. There is a need to optimize many parameters to reach optimal design. Use of thermal-electrical simulation plus sweep of optimization parameters increases computation time exponentially. Park et al., [Park'16], proposed Bayesian Optimization Algorithm, BOA, to optimize the design parameters taking accuracy, speed, scalability into consideration.

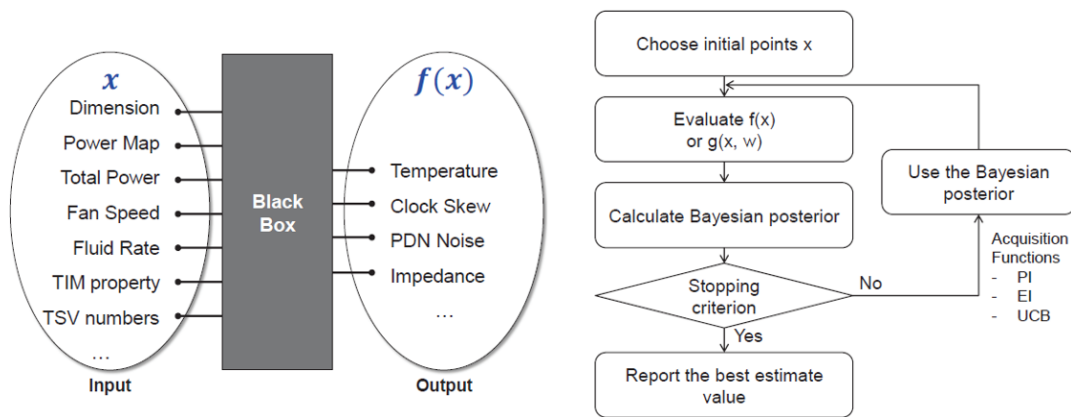


FIGURE 2-1 Using BOA to optimize design parameters [Park'16]

Using full system Thermal Electrical Simulation to run and get few datasets across optimization parameters. Fig. 2-1 shows the extracted features like the dimension, total power, fan speed, TSV number, etc. and the corresponding measured outputs, temperature, clock skew, PDN noise, etc. used to train the model. Now switch to BOA to use this dataset to learn from experience instead of sweeping all parameters in Monte Carlo Analysis. Near real simulation accuracy is achieved with small run and cost time.

3D-IC SW NoC Research: Another 3D-IC ML optimization application Small World NoC where it's needed to transmit data with low latency, high throughput, and minimum power

consumption. 3D-IC NoC enables high performance and low-power many core chips. There is a need for energy-efficient and reliable 3D SW NoC.

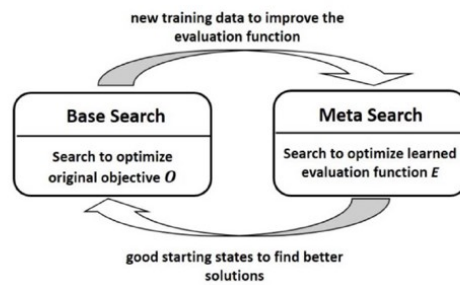
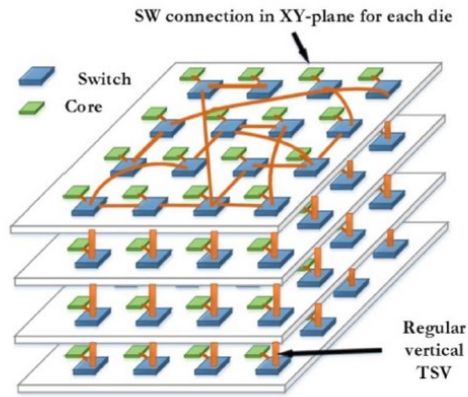


FIGURE 2-2 Small-World NoC energy-efficient and reliable 3D optimization [Das'17]

To achieve this objective, it's needed to optimize placement of planar and vertical communication links and optimize the placement of spare vertical links sVLs in a 3-D NoC. Das et al., [Das'17], proposed using stage learning algorithm, Fig. 2-2, to perform this optimization. Stage starts with base search optimization which generates new training data used to perform a meta search that enhances the optimization process.

2.2 Power Estimation

Power models are either not available or not accurate which results in over or under-estimated power designs. Real HW is not always available. Lee et al., [Lee'15] proposed training a model using real hardware and then calibrate McPAT (Multicore Power, Area, and Timing) simulator,

the open-source simulator used by hpLabs McPAT models. Lee et al' work utilized the least square regression technique.

Power estimation depends on running SW applications on cycle accurate instruction set simulator, ISS, simulation. This process requires lengthy simulation and huge HW resources. Zheng et. Al, [Zheng'15] proposed technique to train a model using cycle accurate simulator results to predict target power estimate while running on a host machine as Fig. 2-3 shows. They used constrained locally sparse linear regression, CLSLR, technique to train the ML model which is a variant of Linear Regression technique.

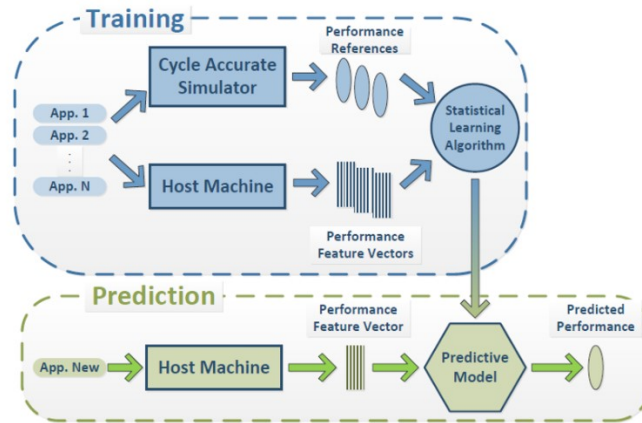


FIGURE 2-3 Power estimation using CLSLR regression technique [Zheng'15]

2.3 Functional Verification

Functional verification of complex designs involves thousands of test cases that require huge time to execute. Most of the time there are redundant test cases that consume time, effort, machine power and don't add any coverage value. Wang et al., [Wang'15], proposed machine learning techniques using SVM, Fig. 2-4, to add a filtering module that performs data mining to skip repeated tests that don't add any coverage and generate unique tests that increase test coverage.

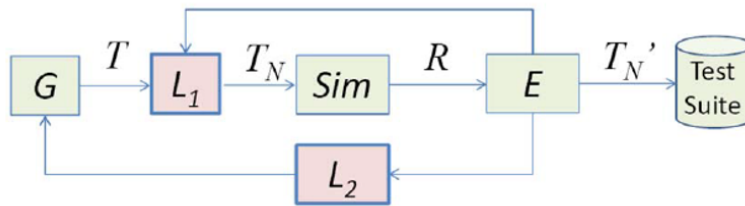


FIGURE 2-4 Functional verification to filter and optimize test cases [Wang'15]

Mandouh et al., [Mandouh'16] focused on the problem of having huge number of assertions, and test failures that have common root causes that causes duplicate effort to debug/fix from large team. Their approach, Fig. 2-5, uses K-Means clustering to extract similar failing coverage groups, assertions trace signals to accelerates debugging and verification of large designs.

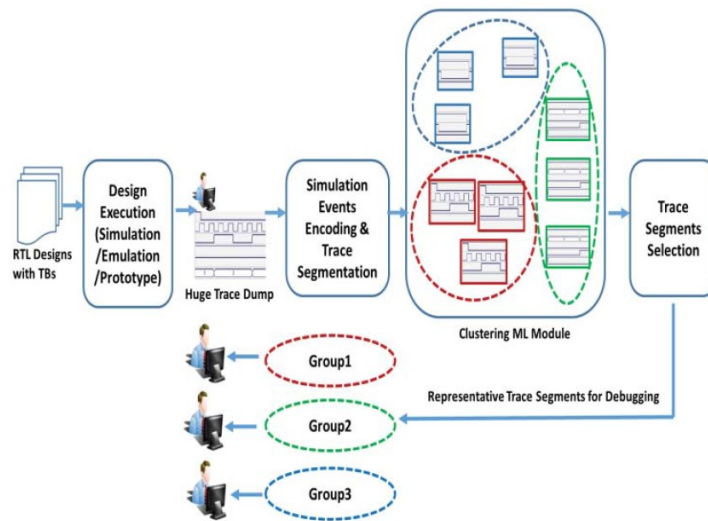


FIGURE 2-5 Grouping of failing coverage groups having common root causes using K-Means clustering [Mandouh'16]

2.4 Timing Analysis

Kahng et al., [Kahng'17], proposed ways to reduce the cost and time of performing signal integrity timing analysis used in Synopsys Prime Time-SI commercial product. Capacitor inductor effects create serious timing errors, shrinking transistor feature size results in huge CrossTalk impact, signal glitches and jitters cannot be ignored as well as noise and distortion. Kahng et al., [Kahng'15], trained a model with SI and non-SI mode timing reports using both

ANN and SVM, Fig. 2-6, and used a weighted prediction of both methods to estimate the SI reports from non-SI reports.

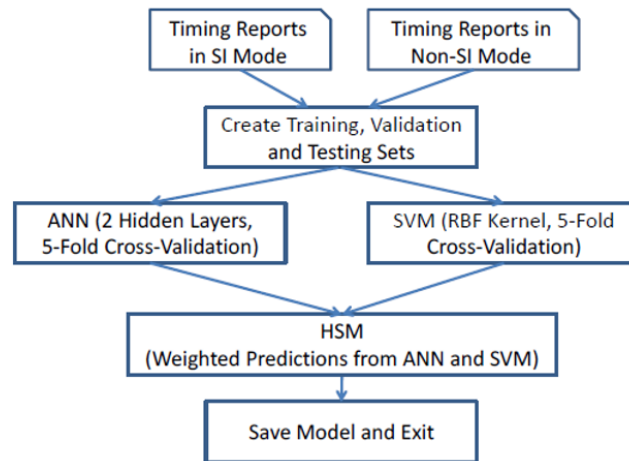


FIGURE 2-6 SI for free proposed model [Kahng'15]

Wang et al., [Wang'17], proposed ML application to refute false positive and false negative critical paths during static timing analysis. There are many reported false negative critical paths in order of thousands. There are also many missed real critical paths in order of hundreds. Wang et al. used machine learning to perform correlation between STA and post-silicon timing measurement using SVM learning techniques to uncover learning rules for improvement.

2.5 Physical Design

Physical proximity of certain mask shapes can lead to short or open circuit which reduces IC manufacturing yield. Different solutions with different limitations exist to address this problem including simulation, design rule checks, DRC and pattern matching. Machine learning provides solutions that can generalize the learnt patterns. Yang et al., [Yang'17] proposed convolutional neural networks to perform hotspot detection using CNN setup in Fig. 2-7.

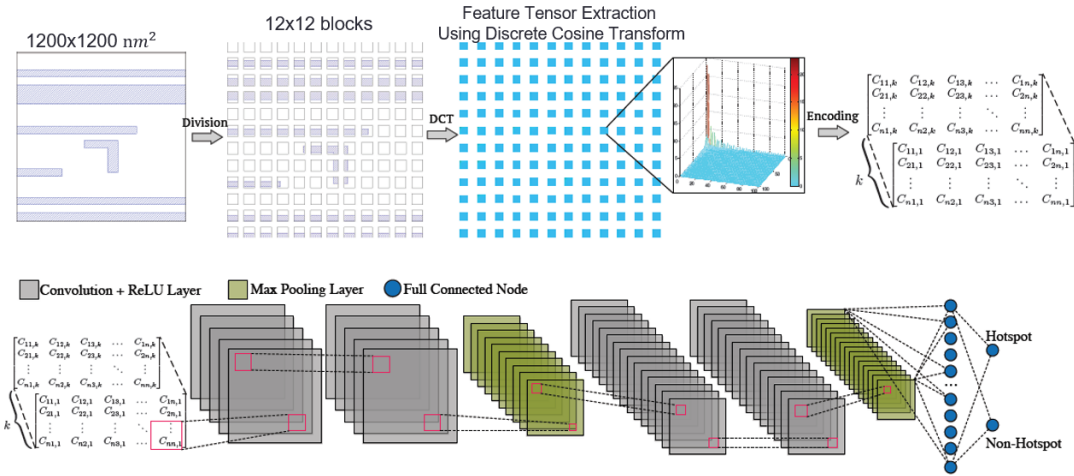


FIGURE 2-7 Hotspot detection using CNN [Yang'17]

Predicting high variability area due to process and voltage variability is another physical design ML application. Drmanac et al., [Drmanac'09], performed feature extraction of variability measures, performed supervised SVM learning to build a variability classifier for inter-cell variability classification. They also performed unsupervised SVM outlier learning to predict areas of high variability.

2.6 Cell Characterization

Cell Characterization is an important domain for machine and deep learning applications. Since these focus on cell delay modeling, cell characterization ML applications will be discussed in next chapter.

2.7 Routing

Meeting IR-drop and EM constraint requires more power distribution network, PDN, metal which increases the overall routing overhead with increased need to speed up design closure process. Chang et al., [Chang'17], proposed using routing cost rather than total area cost and created a learned routing-cost model using Gaussian process regression as in Fig.2-8.

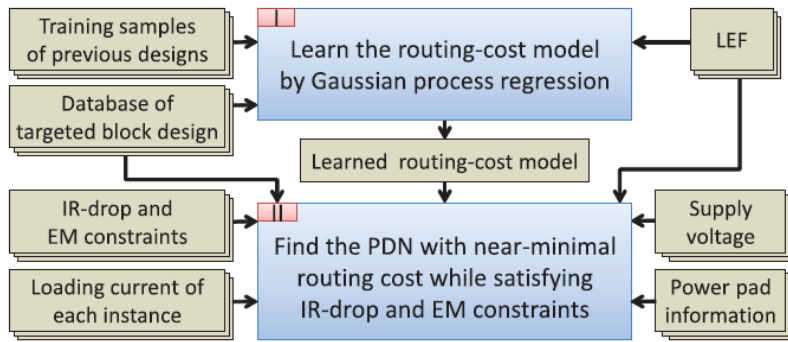


FIGURE 2-8 ML-based routing model using Gaussian process regression [Chang'17]

2.8 Other EDA Applications

Provan et al. [Provan'12] explored machine learning applications in circuit synthesis. Hu et al., [Hu'16], explored machine learning techniques to check equivalence between synthesis and RTL logic. Wu et al., [Wu'17] explored machine learning applications to improve Boolean satisfactory, SAT, solving algorithms. Capodiecici et al., [Capodiecici'17], explored machine learning applications to optimize yield of IC manufacturing. Dai et al., [Dai'17] explored deep learning applications in hardware security problem of circuit recognition. Jap et al., [Jap'16], explored supervised and unsupervised learning technique to detect side-channel based Trojan attacks. Tenace et al., [Tenace'17], used classification and regression tree, CART, to extract the most significant primary inputs and their logic relationships with accuracy that can reach 89%.

Chapter 3

Cell Characterization Models

3.1 Overview

Cell characterization is the process of extracting important cell characteristics to build an accurate and efficient cell model that represents its behavior. Models are used in various EDA digital design flow, [Kahng'11], and tools:

- Static Timing Analysis
- Synthesis
- Power Analysis and Estimation
- Placement and Routing
- Floor Planning
- Automatic Test Pattern Generation, ATPG tools
- Verification Tools

There are several quality metrics to take into consideration while designing and generating these cell models, mainly:

- Accuracy - how far models deviate from silicon behavior
- Conformance - with existing EDA tools
- Completeness - characterizing needed information for time, area, and power
- Efficiency - effort models need at run-time inside EDA tools
- Characterization time - effort needed off-line to characterize cells
- Size of files on disk - how much model information is needed to be stored and maintained

The approach of trading accuracy with run time performance is usually taken to guarantee

adequate cell model that executes fast in different design flows.

In addition, it's needed to characterize each cell against PVT (process, voltage, and temperature) variations, if we have:

- Temp: -40, 25, 80, 125, 150
- Vdd: 0.52, 0.65, 0.8, 0.92, 1.1, 1.3
- Process: FF, SS, TT, FS, SF

Then we have $5 \times 6 \times 5 = 150$ combinations = 150 costly Spice Simulation

3.2 Liberty File Format

Provides lookup tables that store standard cell and process information:

- Rise and fall time of logic cells given input transition time and load capacitance (for NLDM model)
- Output transition time to be used in the next stage of logic
- Setup and hold time for sequential parts
- Characteristics of different tradeoff cell variants:
- High speed, high density, low power, low leakage, low voltage, low noise
- Other important cell characterization information like area and power
- Manufacturing process information, nominal operating temperature, supply voltage variations
- On chip variation information

Library technology is basically the outcome of cell characterization process. Synopsys Liberty .lib file format is the de-facto standard format.

3.3 Liberty File Structure

- Structure Information. Cell connectivity information
- Functional Information. Cell functionality information

- Timing Information. Pin to pin timing information and delay calculations
- Environmental Information. Manufacturing process information, nominal operating temperature, supply voltage variations, etc.

3.4 Non-linear Delay Model, NLDM

The most notable cell driver model is the non-linear delay model as explained by Sharma, [Sharma'16]. The characterization process starts with connecting the cell output pin to a load capacitance and applying a ramp signal to the cell input using a transistor level simulator. The ramp transition time and the load capacitance are swept through defined a range to measure the output delay and transition time.

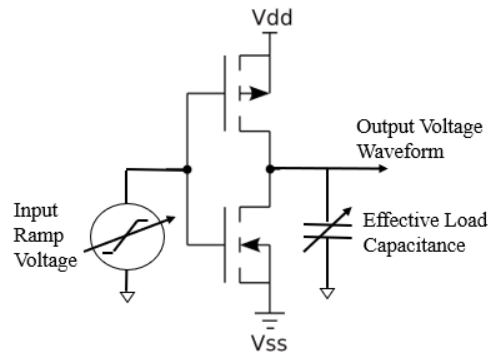


FIGURE 0-1 Cell Characterization Setup

The output delay and transition time parameters at in given input transition and load capacitance are measured and stored in technology files. Usually, the process is repeated twice; one for the rising-edge and the second for the falling-edge. This model is also known as input-Slew, output-Load Delay-Slew Model, SLDSM. Cell characteristics are either:

- Fitted to closed form characteristic equations and stored in the form of k-factor for those equations
- Stored as 2-D look-up table which is the widely used approach. Interpolation is used to retrieve output ramp voltage delay and transition time

The lookup tables are the commonly used method, we proposed DL models in chapter 4 to

replace those traditional tables to increase the error rates without increasing the technology file size. Static timing analysis tools make use of these technology tables to perform its timing analysis. Cell driver model is a voltage source that outputs a ramp-voltage with delay and transition time dependent on input transition and output load values retrieved from the lookup tables or the characteristic equations. Cell receiver model is simply the load capacitance, the effective capacitance, or the PI model of the interconnect.

```

lu_table_template (delay_template) {
  variable_1 : total_output_net_capacitance ;
  variable_2 : input_net_transition ;
  index_1 ("1.0, 1.1, 1.2, 1.3, 1.4");
  index_2 ("2.0, 2.1, 2.2, 2.3, 2.4");

  cell_fall ("delay_template") {
    index_1 ("0.0033500, 0.016, 0.033, 0.067, 0.301") ;
    index_2 ("30.00, 150.00, 500.00, 800.00, 1249.00") ;
    values (\
      "24.00, 48.00, 79.00, 97.00, 118.00",\
      "50.00, 95.00, 159.00, 196.00, 237.00",\
      "79.00, 135.00, 227.00, 280.00, 336.00",\
      "134.00, 201.00, 329.00, 404.00, 485.00",\
      "521.00, 598.00, 799.00, 944.00, 1127.00"\
    );
  }
  cell_rise ("delay template") {
    index_1 ("0.003, 0.016, 0.0335, 0.067, 0.301") ;
    index_2 ("30.00, 150.00, 500.00, 800.00, 1250.00") ;
    values (\
      "26.17, 43.00, 55.00, 56.00, 53.00",\
      "61.00, 99.9352440, 145.9027364, 164.00, 181.00",\
      "103.00, 151.00, 226.00, 260.00, 295.00",\
      "188.00, 242.00, 350.00, 410.00, 473.00",\
      "779.00, 834.00, 991.00, 1114.00, 1268.00"\
    );
  }
}

```

FIGURE 0-2 NLDM Rise/Fall delay time LUT example

```

fall_transition ("delay_template") {
  index_1 ("0.003, 0.016, 0.033, 0.067, 0.301") ;
  index_2 ("30.00, 150.00, 500.00, 800.00, 1249.00") ;
  values (\
    "16.00, 38.00, 82.00, 114.00, 157.00",\
    "42.00, 70.00, 133.00, 177.00, 230.00",\
    "74.00, 103.00, 179.00, 230.00, 293.00",\
    "141.00, 166.00, 253.00, 315.00, 390.00",\
    "622.00, 628.00, 697.00, 770.00, 886.00"\
  );
}
rise_transition ("delay_template") {
  index_1 ("0.003, 0.016, 0.033, 0.067, 0.301") ;
  index_2 ("30.00, 150.00, 500.00, 800.00, 1250.00") ;
  values (\
    "25.00, 52.00, 100.00, 133.00, 180.00",\
    "80.00, 108.00, 181.00, 228.00, 290.00",\
    "152.00, 172.00, 255.00, 316.00, 393.00",\
    "295.00, 306.00, 387.00, 460.00, 555.00",\
    "1299.00, 1299.00, 1323.00, 1389.00, 1483.00"\
  );
}

```

FIGURE 0-3 NLDM Transition LUT example

This ramp model fails short to capture signal high nonlinearity, noise, and crosstalk effects when the feature size decreases, and the interconnect delays increase, and timing analysis becomes more sensitive to signals over-shoot, under-shoot, and multiple crossings. Hence, more complex models were introduced to increase the accuracy of cell models.

3.4 Current Source Model Waveforms Background

3.4.1 Effective Current Source Model, ECSM, Waveforms

In addition to the cell characterization delay and transition-time tables, the output voltage waveform at a given transition time and output load is also stored in the technology file. Waveform information is stored in the output voltage table ECSM format that is introduced by Cadence and adopted as a Silicon Integration Initiative standard, [Si2'21]. The more waveform points we store in the technology file, the more accurate reconstructed waveform we get. However, the accuracy of the stored waveforms comes on the expense of the size of technology file on disk and the performance of static analysis tools making use of these files. Hence, trade-offs between voltage waveform accuracy and data size on disk must be applied.

3.4.2 Composite Current Source, CCS, Waveforms

Composite current source model depends on storing the output current waveform flowing through the capacitive load at given transition input. Waveform information is stored in the output current table CCS format that is introduced by Synopsys, [Synopsys'21]. For current waveforms we only can sample N-points current-time samples at certain time increments. Like ECSM waveforms, the more samples per waveform we take the better waveform accuracy, but on the expense of more size on disk and less performance in static timing analysis. Hence, we must trade current waveform accuracy with data size on disk or find an efficient way to compress CCS and ESCM waveform information.

```

output current rise () {
  vector ( CCS rise ) {
    reference time : 1.018468e+00 ;
    index_1 ( "0.0100" ) ;
    index_2 ( "0.104" ) ;
    index_3 ( "1.026000e+00, 1.030780e+00, 1.030994e+00,
1.034013e+00, 1.213171e+00, 1.316816e+00, 1.447324e+00,
1.584940e+00, 1.715004e+00, 1.868917e+00, 2.044892e+00,
2.377380e+00" ) ;
    values ( "1.277399e-01, 2.053170e-01, 2.381160e-01,
2.377790e-01, 2.142830e-01, 1.894730e-01, 1.310250e-01,
7.119820e-02, 3.582050e-02, 1.474390e-02, 5.060350e-03,
7.094180e-04" ) ;
  }
}

```

FIGURE 0-4 CCS rising-edge current waveform example

And the fall CCS example is:

```

vector ( CCS fall ) {
  reference time : 2.172803e+00 ;
  index_1 ( "0.879602" ) ;
  index_2 ( "0.102702" ) ;
  index_3 ( "1.795857e+00, 2.147012e+00, 2.411707e+00,
2.641051e+00, 3.069397e+00, 3.223209e+00, 3.320412e+00,
3.351926e+00, 3.403433e+00, 3.494074e+00, 3.662858e+00,
3.763292e+00, 3.899033e+00" ) ;
  values ( "0.000000e+00, -5.749690e-03, -1.797790e-02,
-4.025600e-02, -1.030730e-01, -1.222390e-01, -1.298390e-01,
-1.305350e-01, -1.280700e-01, -1.078560e-01, -4.595000e-02,
-2.170950e-02, 0.000000e+00" ) ;
}

```

FIGURE 0-5 CC falling-edge current waveform example

3.5 Current Source Cell Receiver Model

Cell behavior is modeled as voltage-controlled current source so to be independent of the load.

This CSM can model high nonlinearity cell behavior and it can model noise and crosstalk effects by adding and characterizing Miller capacitance, C_m .

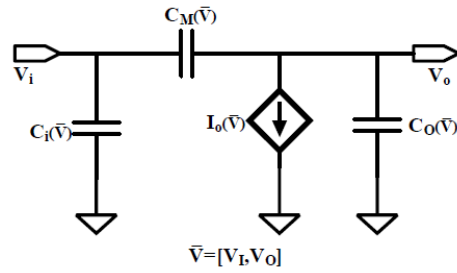


FIGURE 0-6 CSM for simple inverter

For a simple inverter, I_o and C_i , C_m , and C_o are used to characterize the cell given values of V_i and V_o

- I_o is characterized through DC simulations over multiple combinations of DC values of (V_i, V_o) , while C_{int} is characterized through a set of transient simulations
- Interconnect trees are also reduced to PI-model, small circuit simulation is performed with the CSM and the PI-model to get the driving point waveform
- Full interconnect simulation is performed with the driving point waveform to determine the waveform delay and shape at the sink points

NOR2, NAND2 requires 2 current sources and 6 capacitances to model Miller, input, and output capacitances. Number of parameters increases with the complexity of cells; table size increases significantly.

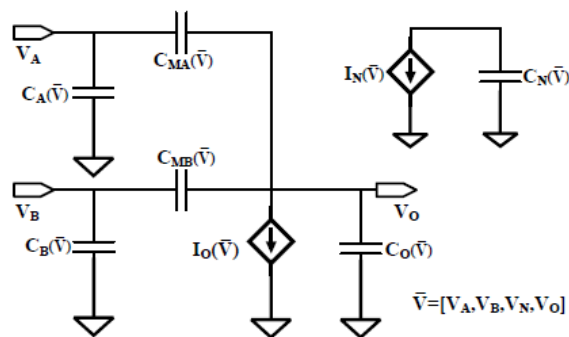


FIGURE 0-7 CSM of a NOR2 Cell

Chapter 4

Deep Learning Autoencoder-based Compression for Current Source Model Waveforms

4.1 Overview

Accounting for larger wire delays, noise and inductance effects on smaller transistor feature sizes resulted in the need to capture more complex waveforms during library cell characterization process. The more complex waveforms, the more waveform samples that need to be stored on disk to capture waveform overshoots, undershoots and multiple crossings. Increased technology file size can have drastic effects on the performance of digital design flow processes and static timing analysis that depend on these library files. Current source model waveform data compression became a necessity to reduce the size of technology files and increase the accuracy of the cell characterization data.

Waveform compression techniques are needed to increase the waveform fidelity without increasing the technology file size. General data compressors can be used like gzip, zlib, and bz2 techniques [Berz'15]. Singular value decomposition, SVD, technique is a linear algebra algorithm to decompose any rectangular $M \times N$ matrix, A , into three matrices, left singular matrix, sigma matrix and right singular matrix. The best sigma matrix rank can be selected and ignore less significant values. The best rank matrix represents the data basis that can represent the compressed version of the original matrix A . SVD technique can decompose any rectangular matrix, and it doesn't require intensive computation power or GPU. Ramalingam et al., [Ramalingam'07], proposed fixed waveform-basis principal component analysis using the

singular value decomposition technique. Hatimi et al., [Hatami'09], used an adaptive waveform-basis singular value decomposition to get better compression results of certain waveforms. They achieved compression ratios between 20 to 4 depending on the accuracy level. Recursive polynomial representation is used by Saurabh and Mittal, [Saurabh'18]. Li and Yu proposed signal-value compression scheme achieving average of 26 compression ratio [Li'21]. Abrishami et al., [Abrishami'19], proposed a neural network simulation framework to eliminate current source model lookup tables without performing waveform data compression to reduce memory latency and computation of current source model. Deep learning Autoencoder, [Goodfellow'16], is a feedforward neural network with multiple hidden layers. Autoencoders are trained to produce an output close to the training input data. The layers from the input layer down to the middle layer, which is typically of a size much smaller than the size of the input data, represent the data encoder. The layers from the middle layer down to the output layer represent the data decoder. Using linear activation functions in the Autoencoder layers produces a comparable performance to the principal component analysis, PCA, technique. On the other hand, using nonlinear activation functions should result in better encoding results. Training Autoencoders requires a large dataset and intensive computation power. However, since the training phase is done offline, this intensive computation power can be tolerated for better performance. Fournier and Aloise, [Fournier'19], compared Autoencoders performance with principal component analysis technique using different image datasets. They showed that PCA technique produces comparable accuracy to Autoencoders results at two orders of magnitude faster performance.

We propose using deep learning nonlinear Autoencoders to compress current source models waveform data. We used SVD technique to compress the same waveforms and compared the trained Autoencoders results with the SVD compression results. We trained different Autoencoder models on 142800 voltage waveforms and another 142800 current waveforms

characterizing rising-edge output NOR, NAND, and INV cells. Different encoding models are trained on similar number of falling-edge waveforms. We achieved effective compression ratio of 104 after encoding 1000-points sampled voltage-time waveforms using two encoding parameters with average correlation coefficient greater than 0.999 and less than 0.5% average percentage error at key waveform points. Compared with SVD compression, Autoencoders gave 1.67x better compression ratio for 1000-points sampled voltage waveforms. We also encoded voltage waveforms at varying time 50, 100, and 150 points. Trained models on 150-points sampled waveforms gave better 1.78x better compression ratio than SVD of rank 16. Encoding more complex 1000-points sampled current-time waveforms resulted in an effective compression ratio of 26 and average correlation coefficient of 0.975 and 0.998 using 8 and 16 encoding parameters respectively. Compared with SVD, Autoencoders achieved 1.7x better compression ratio for complex rising-edge current-time waveforms at model loss of $7.6e-5$ and achieved comparable performance for the falling-edge waveforms.

4.2 Autoencoder-based Waveform Data Compression

We propose using Autoencoders to encode waveforms into very small number of parameters. Those parameters can be decoded again to the original waveform within certain error. The less parameters we can represent a complex waveform is the better in terms of reduced size on disk and higher execution performance. The steps we followed for waveform data compression are:

- Generating ECSM and CCS waveforms.
- Preprocessing waveform data.
- Designing Autoencoder models
- Training Autoencoder models.

4.2.1 Generating ECSM and CCS Waveforms

MOS transistor model BSIM3v3 model (Eldo level 53) is used to perform transistor level

simulation of NOR, NAND and INV cells. Each cell is connected to a capacitive load and input ramp voltage source. Using the simple cell characterization setup in Fig. 4-1 for INV, NOR and NAND cells, the transition time of the ramp input is varied from 10ps to 2 ns, whereas the capacitive load is varied from 0.1fF to 100fF.

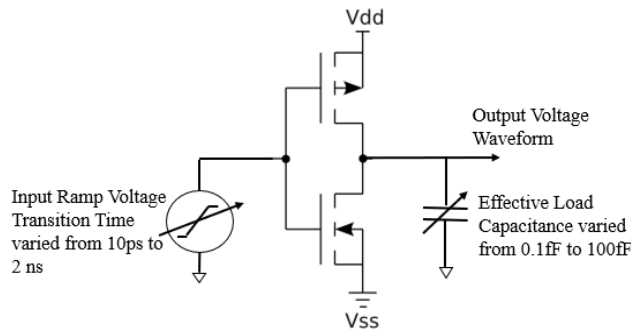


FIGURE 4.1 Cell Characterization Transistor Level Simulation Setup

In each simulation run, the 1000-points voltage-time samples are measured at the output pin to store the waveforms needed for ECSM models. Uniform time 1000 sampled points per waveform, as in Fig. 4-2, is selected to capture more complex waveform properties like overshoots, undershoots, rapid transitions and multiple crossings.

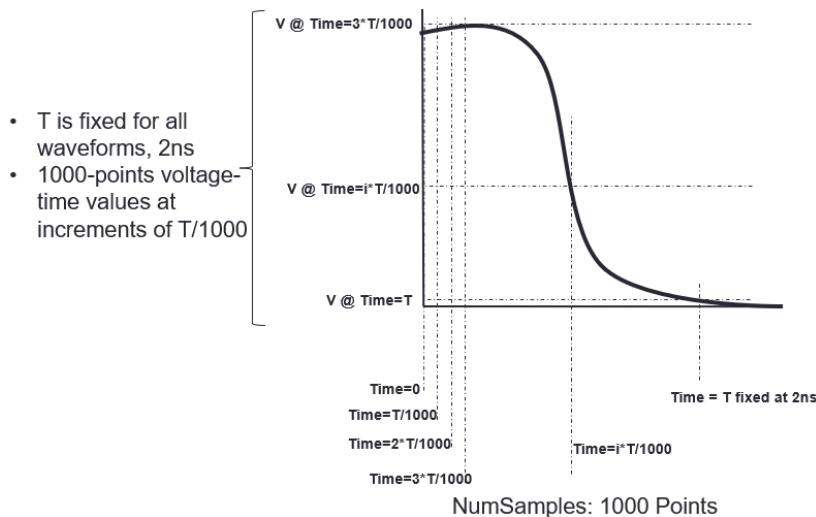


FIGURE 4.2 1000-points uniform time sampled waveform

Another sampling technique is devised to sample the most significant N samples before crossing 95%VDD or 5%VDD for rising or falling time waveforms respectively as shown in Fig. 4-3. The

number of samples N is varied between 50, 100 and 150 points. Sampling 150-points gave better results than 50 and 100, therefore sampling results of 150-points are reported and compared with sampling uniform time 1000-points in this paper.

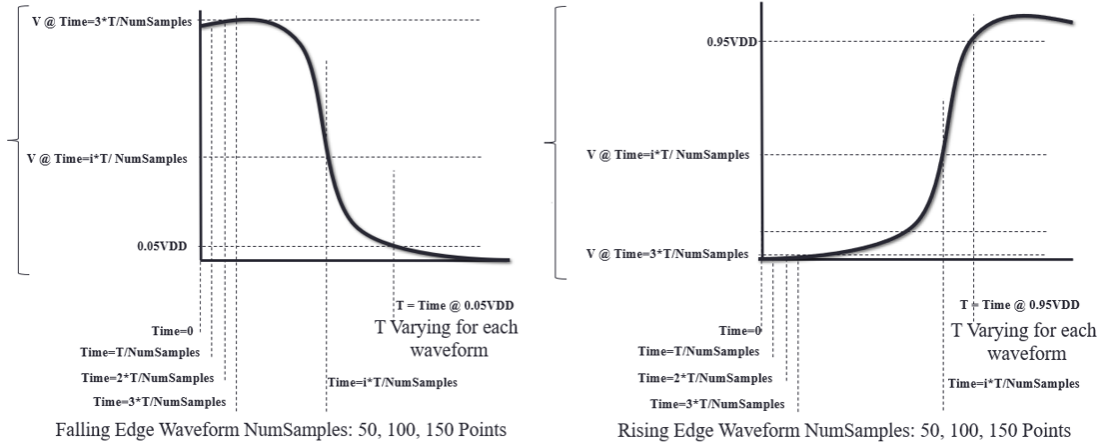


FIGURE 4-3 Varying Time, 50, 100, 150 Sampling Points

Total of 142800 rising-edge and another 142800 falling-edge voltage waveforms were captured for ECSM models. During the same simulation run, delay and transition time values are recorded against the input waveform transition time and effective load capacitance to train the NLDM deep learning models.

4.2.2 Preprocessing Waveform Data

Data normalization is a key step to perform before training deep learning networks. Preprocessing voltage-time waveform data is a little bit more involved to account for the over and under shoots. Assuming 50% possible overshoot or undershoot, then equation (4) normalizes waveform data between 0 and 1 including possible over or under-shoots.

$$\text{Normalized } WF = (WF + 0.5VDD) / (2 * VDD) \quad (4)$$

Current waveforms are more complex to normalize between 0 and 1 because the minimum and maximum current values are different from waveform to the other. Hence, we scaled each

waveform to have similar minimum and maximum values, then normalized all waveform data between 0 and 1.

4.2.3 Designing Autoencoder Models

We designed several Autoencoder models and trained them using the preprocessed ECSM and CCS waveform data. The model topologies that resulted in best training performance are listed in table 4.1. Each model has an input layer I, and output layer O of the same size that is equal to the number of waveform sample points. For each model, the number of the encoded parameters P is varied between 8, 4, 2, 1 for ECSM waveforms and between 16, 8, 4, 2, 1 for CCS waveforms. Relu activation function is used for all Autoencoder layers, and sigmoid function is used in the output layer. Using sigmoid function adds a non-linearity learning capability to the Autoencoders that enables them to learn more complex data.

Table 4.1 Autoencoder Models for Parameter P = 16,8,4,2,1

ID	Autoencoder Models	
	Model Topology	Activation Function
13	I,2048,512, P,512,2048,O	Relu for all layers, except Sigmoid for middle and last layer
12	I,2048,512,16, P,16,512,2048,O	
10	I,2048,1024,32,16, P,16,32,1024,2048,O	
9	I,2048,1024,128,64,32,16, P,16,32,64,128,1024,2048,O	
4	I,1024,64,32,16, P,16,32,64,1024,O	
3	I,1024,32,16, P,16,32,1024,O	
2	I,1024,16, P,16,1024,O	

To calculate the compression ratio, we need to calculate the size of encoded data plus the decoder size to compare with the original data size on disk. Given S number of waveforms, P number of encoded parameters, B number of bytes to store the encoded parameters, then the encoded waveform size, E, is given by equation (5). On the other hand, given R number of actual data points per waveform saved on disk; R=21 data points for ECSM and R=1000 data points for each ECSM or CCS waveform, then the uncompressed data size, O, is given by equation (6). Considering that the total size needed to store the encoded data on disk is the encoded waveform

size plus the decoder size, D , needed to decode the waveforms. Hence, the effective compression ratio is calculated by equation (7).

$$E \text{ (Encoded Waveform Size)} = (S * P * B) \quad (5)$$

$$O \text{ (Original Waveform Size)} = (S * R * B) \quad (6)$$

$$\text{Effective Compression Ratio} = O / (E + D) \quad (7)$$

4.2.4 Training Autoencoder Models

All Autoencoder models with all possible values of the parameter P (16, 8, 4, 2, 1) were trained on the normalized WF data. The used computer configuration is: Intel Core I7, 16GB RAM, with 6GB NVIDIA GeForce RTX 2060 GPU. Python version 3.8.5 and Keras library v2.4 with GPU-enabled Tensorflow backend is used to model Autoencoder models. Adam optimization is used in the training process and mean squared error, MSE, is used as a loss function. The training process was designed to abort training if the loss value deteriorates 50 iterations in sequence. Single Autoencoder model is trained on a combined input of NOR, NAND, INV rising or falling-edge waveforms. Successful training of 142800 1000-points current waveforms took average of 400 epochs in average of 2000 sec. Training of 142800 1000-points voltage waveforms took average of 300 epochs in an average of 1500 sec.

4.3 Waveform Data Compression Results

To evaluate the performance of the trained Autoencoders, all the normalized waveforms were compressed using Python implementation of SVD technique. All the normalized waveforms are packed in a matrix A , of size 142800×1000 or 142800×150 depending on the sampling points. The matrix A is analyzed using SVD. Different ranks of the sigma matrix are selected to produce the compressed version of matrix A . The SVD data compression at different ranks are evaluated and compared to our proposed Autoencoder-based compression at different number of encoding parameters.

4.3.1 Autoencoding fixed time 1000-points ECSM Voltage Waveforms

To evaluate the performance of the trained Autoencoders, all the normalized waveforms were compressed using Python implementation of gzip, bz2 techniques and SVD. Autoencoders of 2 parameters gave between 39x to 45x better compression ratios than gzip and bz2 techniques at less degree of accuracy. For SVD, all the normalized waveforms are packed in a matrix A, of size 142800 times the number of sampling points). The matrix A is analyzed using SVD. Different ranks of the sigma matrix are selected to produce the compressed version of matrix A. The SVD data compression at different ranks are evaluated and compared to our proposed Autoencoder-based compression at different number of encoding parameters.

Table 4.2 Best Autoencoding 1000-points ECSM Falling-Edge Waveform Results Against bz2, gzip and SVD techniques

	Falling-Edge WF										
	bz2	gzip	SVD Rank=16			Autoencoder, Model ID=2					
						2 param			4 param		
% error			Avg	Std	Max	Avg	Std	Max	Avg	Std	Max
@ 0.8VDD	0	0	0.84	2.5	99.9	0.67	1.3	34.7	0.37	0.5	8.1
@ 0.5VDD			0.58	2.1	83.2	0.52	0.8	8.4	0.27	0.3	4.7
@ 0.2VDD			1.10	2.4	84.8	0.66	1.1	34.9	0.29	0.5	5.3
Mean corcoef	1	1	0.99876			0.9995			0.99988		
Loss (MSE)	0	0	8.29E-05			5.2E-05			1.27E-05		
Disk size-MB	188.5	161.6	6.9			4.15			4.15		
Compression Ratio	2.29	2.67	62.5			104.06			86.14		
Compression time (s)	49.6s	32s	0.974s			4.6s			4.6s		
Decompression time (s)	16.2s	10s	0.212s			2.6s			2.6s		

Table 4.3 Best Autoencoding 1000-points ECSM Rising-Edge Waveform Results Against bz2, gzip and SVD techniques

	Rising-Edge WF										
	bz2	gzip	SVD Rank=16			Autoencoder, Model ID=2					
						2 param			4 param		
% error			Avg	Std	Max	Avg	Std	Max	Avg	Std	Max
@ 0.8VDD	0	0	2.9	6.7	210	0.7	1.4	30.3	0.3	0.7	12.6
@ 0.5VDD			1.1	5.7	192	0.46	0.8	30.4	0.25	0.5	7.6
@ 0.2VDD			2.1	5.6	158	0.67	1.5	31.4	0.32	0.7	16.2
Mean corcoef	1	1	0.99535			0.9994			0.99985		
Loss (MSE)	0	0	3.2E-04			5.8E-05			1.37E-05		
Disk size-MB	188.5	161.6	6.9			4.15			4.15		
Compression Ratio	2.36	2.71	62.5			104.06			86.14		
Compression time (s)	49.6s	32s	0.974s			4.6s			4.6s		
Decompression time (s)	16.2s	10s	0.212s			2.6s			2.6s		

Tables 4-2 and 4.3 show that Autoencoders of 2 parameters were able to produce 1.67x better compression ratio results than SVD technique of rank 16, and less than 0.7% average percentage error compared to SPICE simulation. The tables show also that encoding with 4-params

outperforms 2-params encoding by $\sim 2x$ factor. The standard deviation of 2-params encoding is less than 1.5% for key waveform parameters. The offline training time is 600s for 100 epochs it can be enhanced with better hardware resources and smaller encoding models. Autoencoders of 4 parameters gives better average and maximum percentage errors at less compression ratio compared to compressed 2 parameters.

4.3.2 Autoencoding varying time ECSM Voltage Waveforms

The same experiment is repeated with 50,100 and 150-points sampled waveforms at varying time range as described in section 4.2.1. Tables 4.4 and 4.5 show that Autoencoders ID 17 gave best results for 2-4 params encoding of rising and falling edges. Autoencoding with 2 parameters give 1.79x better compression ratio results than the nearest SVD model of rank 4 with more than $\sim 1.5x$ better average, maximum and standard deviation of percentage errors. Autoencoders of 2 parameters also gave between 40x to 55x better compression ratios than gzip and bz2 techniques at less degree of accuracy. This accuracy is achieved with offline training time of 100s for 100 epochs. Encoding with 4-params resulted in $\sim 2x$ better results than encoding with 2-params. The encoded 150-points sampled waveforms remain overall better than the encoded 1000-points sampled waveforms in average and maximum percentage errors by $\sim 2x$ factor. Fig. 4-4 shows a sample falling and rising edge waveform reconstructed after being encoded.

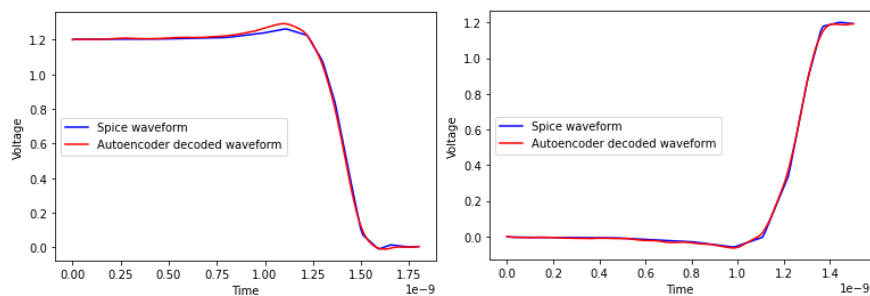


FIGURE 4-4 Falling and rising decoded waveforms against Spice waveforms

Table 4.4 Best Autoencoding 150-points ECSM Waveform Results Against bz2, gzip and SVD techniques

	Falling-Edge WF										
	bz2	gzip	SVD Rank=4			Autoencoder, Model ID=17					
						2 param			4 param		
% error			Avg	Std	Max	Avg	Std	Max	Avg	Std	Max
@ 0.8VDD	0	0	0.82	1.9	32.8	0.39	0.66	8.7	0.19	0.3	3.43
@ 0.5VDD			0.42	0.65	13.7	0.29	0.54	9.37	0.12	0.2	3.36
@ 0.2VDD			0.61	1.4	251.8	0.49	0.85	12.0	0.12	0.2	3.99
Mean corrccoef	1	1	0.99632			0.99911			0.99978		
Loss (MSE)	0	0	1.78E-04			5.1E-05			9.01E-06		
Disk size-KB	6805	5501	245			136			136		
Compression Ratio	1.35	1.67	37.5			67.45			35.35		
Compression time (s)	16s	2.4s	0.06s			0.62s			0.62s		
Decompression time (s)	1.45s	0.24s	0.03s			0.52s			0.52s		

Table 4.5 Best Autoencoding 150-points ECSM Waveform Results Against bz2, gzip and SVD techniques

	Rising-Edge WF										
	bz2	gzip	SVD Rank=4			Autoencoder, Model ID=17					
						2 param			4 param		
% error			Avg	Std	Max	Avg	Std	Max	Avg	Std	Max
@ 0.8VDD	0	0	0.47	0.84	27.6	0.38	0.7	11.97	0.15	0.2	3.49
@ 0.5VDD			0.59	1.3	35.96	0.27	0.4	7.08	0.12	0.2	3.09
@ 0.2VDD			0.65	1.9	43.08	0.37	0.7	5.44	0.22	0.4	4.92
Mean corrccoef	1	1	0.99289			0.99854			0.99973		
Loss (MSE)	0	0	2.15E-04			7.27E-05			1.17E-05		
Disk size-KB	7593	6048	245			136			136		
Compression Ratio	1.21	1.51	37.5			66.85			35.35		
Compression time (s)	7.1s	1.2s	0.06s			0.62s			0.62s		
Decompression time (s)	1.45s	0.24s	0.03s			0.52s			0.52s		

Tables 4.6 and 4.7 compare 2 and 4 parameters Autoencoder results for 1000-points, and varying time 50, 100, 150 points sampled waveforms.

Table 4.6 Autoencoding 50,100,150,1000-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Falling-Edge WF	Autoencoder Number of Encoding Parameters							
	4				2			
Num of Params	50	100	150	1000	50	100	150	1000
Num of waveform samples	50	100	150	1000	50	100	150	1000
Model ID	15	17	16	2	15	14	16	2
Decoder size (KB)	39	111	125	4249	39	33	125	4249
Avg % error 0.8VDD	1.8758	0.8707	0.19	0.37	2.316	1.3110	0.39	0.7
Avg % error 0.5VDD	1.861	0.9338	0.12	0.27	2.1428	0.8739	0.29	0.46
Avg % error 0.2VDD	1.8777	0.9652	0.12	0.29	2.0736	1.0262	0.49	0.67
Mean corrccoef	0.99982	0.99994	0.99978	0.99985	0.9996	0.99948	0.99911	0.9995
Model Loss (MSE)	1.24E-05	6.18E-06	9.01E-06	1.37E-05	3.87E-05	6.18E-05	5.1E-05	5.2E-05
Compression Ratio	12.28	23.8	35.51	86.13	24.15	48.58	67.45	104.06

Table 4.7 Autoencoding 50,100,150,1000-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Rising-Edge WF	Autoencoder Number of Encoding Parameters							
	4				2			
Num of Params	50	100	150	1000	50	100	150	1000
Num of waveform samples								
Model ID	17	17	16	2	15	15	17	2
Decoder size (KB)	86	86	125	4249	39	51	136	4249
Avg % error 0.8VDD	2.0015	0.9643	0.15	0.3	2.1108	1.1525	0.38	0.7
Avg % error 0.5VDD	1.9264	0.9889	0.12	0.25	2.0596	0.9409	0.27	0.46
Avg % error 0.2VDD	1.9350	1.0649	0.22	0.32	2.1574	1.2043	0.37	0.67
Mean corrcoeff	0.99991	0.99992	0.99973	0.99985	0.99966	0.99954	0.99911	0.9994
Model Loss (MSE)	4.96E-06	6.03E-06	1.17E-05	1.37E-05	3.00E-05	5.01E-05	5.1E-05	5.8E-05
Compression Ratio	12.03	23.81	35.51	86.13	24.15	47.81	66.85	104.06

4.3.3 Autoencoding fixed time 1000-points CCS Current Waveforms

CCS current Waveforms are more complex than the ECSM voltage waveforms, hence it's expected to require more parameters to achieve acceptable accuracy results. Tables 4.8 and 4.10 list the best Autoencoder models at different number of parameters for falling and rising edge waveforms respectively. Autoencoders with single parameter didn't converge, so they are omitted from the tables. Falling edge Autoencoder gave better results at 8 encoding parameters for model ID#10, and 26.8 effective compression ratio. Table 4.9 shows the SVD compression results of the same waveforms. At Sigma of rank 32, SVD gives 1.15x better compression results than the nearest Autoencoder model of 8 encoding parameters. However, SVD runtime decoding can be 19x faster than Autoencoders decode time. Fig. 4-5 shows a sample reconstructed 8-parameters encoded falling-edge waveform.

Table 4.8 Autoencoding 1000-points Falling-Edge CCS Waveform Results at Different Number of Encoding Parameters

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	12	10	9	10
Decoder size (KB)	12163	16368	16795	16368
Mean corrcoeff	0.95310	0.97501	0.94947	0.9365
Model Loss (MSE)	6.20E-04	5.90E-04	8.00E-04	5.7E-03
Compression Ratio	26.46	26.79	29.34	31.93

Table 4.9 SVD 1000-points Falling-Edge CCS Waveform Results at Different Sigma Rank Number

Falling-Edge WF	SVD Sigma Rank Number			
	64	32	16	8
Mean corrccoef	0.99666	0.98393	0.94397	0.82042
Model Loss (MSE)	1.47E-04	7.00E-04	2.60E-03	7.9E-03
Compression Ratio	15.41	30.81	61.63	123.27

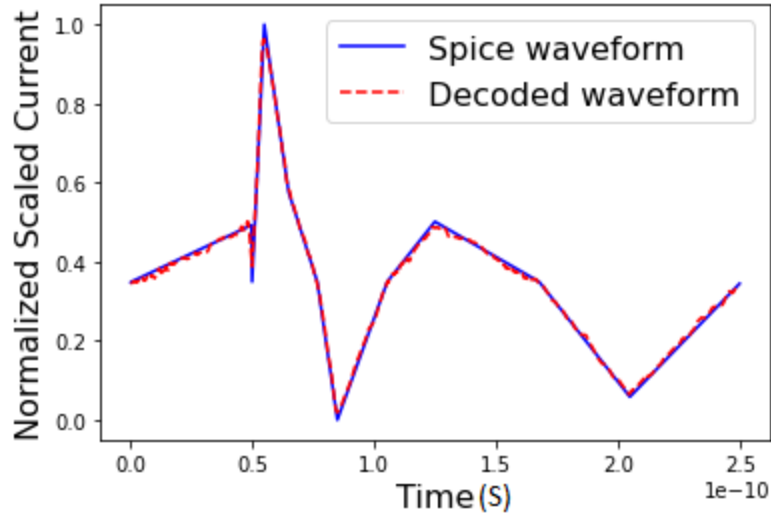


FIGURE 4-5 DECODED 8-P CCS FALLING-EDGE NORMALIZED AND SCALED CURRENT WAVEFORM VS. SPICE WAVEFORM

Rising-edge current waveforms gave better results at 16 encoding parameters, model ID#13 and compression ratio of 26.5. Table 4.11 shows the SVD compression results of the same waveforms. In this case, Autoencoder of 16-parameters gave 1.72x better compression ratio than the nearest SVD Sigma matrix of rank 64. Fig. 4-6 shows a sample reconstructed 16-parameters encoded rising-edge current waveform.

Table 4.10 Autoencoding 1000-points Rising-Edge CCS Waveform Results at Different Number of Encoding Parameters

Rising-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	13	12	2	3
Decoder size (KB)	12128	12163	4249	4320
Mean corrccoef	0.99846	0.99765	0.97981	0.94083
Model Loss (MSE)	7.60E-05	1.10E-04	1.10E-03	3.5E-03
Compression Ratio	26.51	33.57	86.13	102.7

Table 4.11 SVD 1000-points Rising-Edge CCS Waveform Results at Different Sigma Rank Number

Rising-Edge WF	SVD Sigma Rank Number			
	128	64	32	16
Mean corrccoef	0.96587	0.93526	0.90055	0.70785
Model Loss (MSE)	2.33E-05	7.99E-05	3.44E-04	1.3E-03
Compression Ratio	7.71	15.41	30.81	61.63

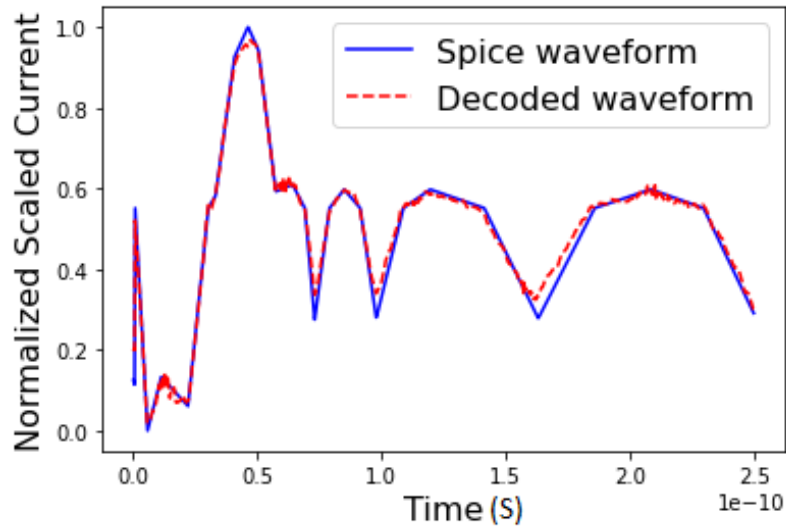


FIGURE 4-6 DECODED 16-P CCS RISING-EDGE NORMALIZED AND SCALED CURRENT WAVEFORM VS. SPICE WAVEFORM

4.4 Conclusions

Encoding 1000-points sampled waveforms with 2-params outperforms the nearest SVD 16-rank compression ratio by 1.67x and the generic lossless compression ratio by ~45x. It also outperforms the technology file LUT by ~3x reduction in storage (assuming storing 8 points per waveform and storing all number of waveforms). Encoding with 4-params outperforms the 2-params encoding waveform quality by ~2x, and the nearest SVD rank compression ratio by 1.37x. Similarly, 4-params encoding outperforms the technology file LUT by ~4x reduction in storage (assuming storing 8 points per waveform and storing all number of waveforms). Encoding 150-points waveforms outperformed the 1000-points waveform encoding by 2x. However, the nearest

SVD 16-rank outperforms 2-params encoding decompression time by $\sim 12.25x$ for 1000-points and by $17.34x$ for 150-points waveforms. There is a room for improving the reported decompression time of the Autoencoders either through finding better DL models with smaller size, or by using ML accelerated hardware. In the following chapter, these encoded waveform parameters will be used in two proposed models; one that replaces both NLDM-LUT and ECSM waveforms using a combined model, and another DL model to propagate waveform parameters instead of depending only on the waveform transition time.

Chapter 5

Deep Learning Cell Driver Delay Modeling

5.1 Overview

Several deep learning techniques to model cell delays are proposed in this chapter. First, a deep learning non-linear delay model, DL-NLDM, is proposed to replace the traditional non-linear delay model lookup tables, NLDM-LUT, to produce higher degrees of accuracy over a wide range of input parameters. These DL-NLDM models are trained using measured cell output transition and delay time at different input transition time and capacitive load values. The deep learning model ability to generalize allowed achieving better results than lookup tables. Building on chapter 4 waveform Autoencoding technique, the input/output encoded waveform parameters are then used together with the associated effective capacitance to train another DL model and generalize the cell behavior. The trained DL cell model can then be utilized to produce the expected cell output encoded waveform given an encoded input. At any point of time, the cell output can be decoded back to its time-domain form using the corresponding waveform decoder produced earlier during the training phase. Finally, deep learning waveform delay model, DL-WFDM, is proposed instead of NLDM-LUT or DL-NLDM models. In DL-WFDM, auto-encoded input/output waveforms parameters are proposed to model cell delays instead of the cell delay and the transition time used in the NLDM. SPICE-simulated cell input and output waveforms are used to train DL waveform Autoencoders. These trained Autoencoders can encode and decode the whole waveform into few parameters. The input/output encoded waveform parameters are then used together with the associated effective capacitance to train another DL model and generalize the cell behavior. The trained DL-WFDM model can then be utilized to produce the

expected cell output encoded waveform given an encoded input. At any point of time, the cell output can be decoded back to its time-domain form using the corresponding waveform decoder produced earlier during the training phase.

The main contributions of this research work are:

- Modeling cell delays using DL neural networks. This approach reduced the two tables characterizing the cell delay and transition time to one DL model. The DL-NLDM performed better than the standard 7x7 LUT size in average percentage errors and outperformed the non-standard 100x100 LUT in maximum percentage errors over the selected wide range of input values, and less than 1.4% average percentage delay error compared with SPICE simulation.
- Compressing ECSM voltage time waveforms to increase the waveform fidelity while reducing the technology file size using DL Autoencoders technique. This DL compression technique outperformed the SVD technique by 1.67x for 1000-points sampled waveforms, and by 1.79x for 150-points sampled waveforms. Autoencoder technique outperformed gzip and bz2 techniques compression by 45x.
- Modeling a combined DL-NLDM-ECSM model trained on two-parameters encoded output waveforms given input transition time and capacitive load values. Instead of using only the delay and transition time to characterize the cell, this approach uses two parameters to represent a complex waveform. Combined models are still better than NLDM-LUT delay and transition time maximum percentage error, however separate DL-NLDM and ECSM compression models still perform better than the combined models.
- Modeling cells using DL-WFDM model trained on two-parameters encoded input waveforms plus capacitive load values as inputs and two-parameters encoded output waveforms. The results of the DL-WFDM are promising, but separate DL-NLDM and

ECSM compression models still perform better than these new models.

5.2 DL-NLDM: Deep Learning NLDM Cell Delay Model

The same cell characterization setup used to obtain the ECSM voltage waveforms are used to get the training data for NLDM models. For each waveform output, the cell delay is measured as well as the transition time from 80% from/to 20% of VDD for the output voltage. The training data is formed using the input waveform transition time and effective load capacitance, and the output cell delay and the output waveform transition time. Feed-forward fully connected, FC, neural network is constructed and trained using this training data. Fifteen different deep learning model topologies are examined, one DL model and one training process is required for each of NAND, INV, and NOR.

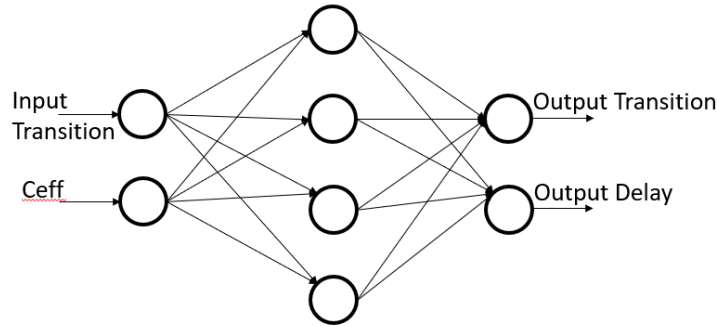


FIGURE 5.1 DL-NLDM cell delay model

Table 5.1 shows the models that gave best modeling results.

Table 5.1 Deep Learning Model Structure

Model ID	FC Neural Network Structure	Activation Functions
2	1,50,25,2	Relu for all layers, Sigmoid for output layer I=2 or 3
3	1,100,2	
4	1,100,50,2	
6	1,200,100,25,2	
8	1,150,75,32,16,2	

The error rates of the deep-learning models are compared with the error rates of NLDM lookup tables, LUT, of different sizes. Though the de-facto size of NLDM LUT is 5x5 or 7x7 tables, DL-NLDM results are compared with other larger non-standard LUT sizes as shown in Tables 5.2,

5.3 and 5.4 for different cell types. To increase the accuracy of the LUT tables, two sets of tables are used for each cell/edge type; one LUT table for input ramp range of 10ps to 1 ns and capacitive load range of 0.1fF to 1fF, and another LUT table for input ramp range of 1ns to 2ns and capacitive load range of 1fF to 100fF. On the other hand, a single DL-NLDM is trained on the whole input range. Offline training time is 20s, retrieving time is almost the same for different cell types, so it's reported for NOR type only in table 5.2.

Table 5.2 DL-NLDM best trained NOR models Mean/Max Percentage Error rates versus different LUT sizes

NOR Falling Edge												
Delay Model	LUT 7x7			LUT 25x25			LUT 100x100			DL-NLDM, Model ID=6		
% error	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
Delay Time	0.53	0.91	15.46	0.22	0.67	15.46	0.16	0.86	18.1	0.48	0.57	3.56
Tr Time	1.65	2.09	33.49	0.89	1.77	33.49	0.51	2.27	52.8	1.19	2.06	8.00
MSE	-			-			-			1.76E-05		
Disk size-KB	2			25.2			408			109		
Retrieving time (s)	0.18s			0.26s			1.28s			0.106s		
NOR Rising Edge												
Delay Model	LUT 7x7			LUT 25x25			LUT 100x100			DL-NLDM, Model ID=6		
Delay Time	0.91	1.21	16.54	0.34	0.82	16.54	0.19	0.94	19.0	0.71	0.93	5.95
Tr Time	4.24	5.44	43.85	2.12	3.99	48.06	0.91	3.63	89.2	3.20	4.72	36.33
MSE	-			-			-			6.29E-05		
Disk size-KB	2			25.2			408			109		
Retrieving time (s)	0.18s			0.26s			1.28s			0.106s		

Table 5.3 DL-NLDM best trained INV models Mean/Max Percentage Error rates versus different LUT sizes

INV Falling Edge												
Delay Model	LUT 7x7			LUT 25x25			LUT 100x100			DL-NLDM, Model ID=6		
% error	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
Delay Time	0.70	0.71	26.31	0.37	1.21	26.31	0.29	1.54	33.1	0.60	0.94	10.41
Tr Time	2.67	3.76	39.85	1.42	3.01	41.58	0.62	2.75	59.8	1.91	3.00	22.67
MSE	-			-			-			3.48E-05		
Disk size-KB	2			25.2			408			109		
Retrieving time (s)	0.18s			0.26s			1.28s			0.106s		
INV Rising Edge												
Delay Model	LUT 7x7			LUT 25x25			LUT 100x100			DL-NLDM, Model ID=6		
Delay Time	2.82	3.72	51.53	0.97	2.12	31.51	0.40	0.40	43.4	1.41	2.67	42.25
Tr Time	9.25	12.13	128.6	3.94	2.77	153.7	1.0	3.86	142.5	4.86	4.86	51.80
MSE	-			-			-			2.35E-04		
Disk size-KB	2			25.2			408			109		
Retrieving time (s)	0.18s			0.26s			1.28s			0.106s		

Table 5.4 DL-NLDM best trained NAND models Mean/Max Percentage Error rates versus different LUT sizes

NAND Falling Edge												
Delay Model	LUT 7x7			LUT 25x25			LUT 100x100			DL-NLDM, Model ID=6		
% error	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
Delay Time	0.61	1.28	23.44	0.26	1.01	23.44	0.24	1.36	28.6	0.49	0.79	7.84
Tr Time	1.32	2.11	38.10	0.65	1.79	38.1	0.46	2.33	50.8	0.85	1.21	7.51
MSE	-			-			-			2.91E-05		
Disk size-KB	2			25.2			408			109		
Retrieving time (s)	0.18s			0.26s			1.28s			0.106s		
NAND Rising Edge												
Delay Model	LUT 7x7			LUT 25x25			LUT 100x100			DL-NLDM, Model ID=6		
Delay Time	1.88	2.58	27.38	0.68	1.54	22.97	0.29	0.29	27.9	0.79	1.34	12.30
Tr Time	5.54	6.13	57.95	2.29	4.51	110.5	0.78	2.86	54.5	2.69	4.56	74.66
MSE	-			-			-			8.66E-04		
Disk size-KB	2			25.2			408			109		
Retrieving time (s)	0.18s			0.26s			1.28s			0.106s		

The above tables show that the single DL-NLDM outperforms the rising and falling edge 7x7 dual-table LUT average error accuracy and outperforms 100x100 dual-table LUT in maximum percentage errors for both delay and transition time over the selected wide range of transition time and effective capacitance values. Table 5.2 shows also that the retrieving time of DL-NLDM is better than those of different NLDM-LUT sizes. The increased accuracy and better retrieving time come at the expense of offline DL-NLDM training time and the increased size of disk, however the DL-NLDM disk size is better than 100x100 LUT size.

5.3 Combined DL-NLDM ECSM Waveform Cell Delay Model

Combined DL NLDM-ECSM deep learning model is proposed to be used to retrieve both NLDM delay/transition time values and the ECSM voltage waveform using a single model per cell type per waveform output edge. The ECSM output waveforms are encoded using the corresponding Autoencoder model encoder to produce the expected output waveform parameters. The combined DL model is trained on input waveform transition time and effective load capacitance values as input to produce the expected output waveform encoded parameters as well as the output transition and delay times as shown in Fig. 5.2. Tables 5.5 to table 5.10 show model ID#8 results which gives the best training results for 1000-points based waveforms. These combined models achieved less than 2.89% average percentage error for key waveform points, less than

1.94% average percentage error for delay time, and less than 6.64% average percentage error for transition time compared with SPICE simulation. Combined NLDM using 150-points varying time encoded waveforms gives better results than 1000-points sampled waveforms with model ID#6 for the average and maximum waveform percentage errors as well as for the average transition time as shown in tables 5.5 to table 5.10. All trained models have 218 KB size on disk. Tables 5.5 to table 5.10 show that separate DL-NLDM with 1000-points and 150-points 2-params encoded ECSM waveforms are compared against combined 1000-points and 150-points DL-NLDM ECSM models.

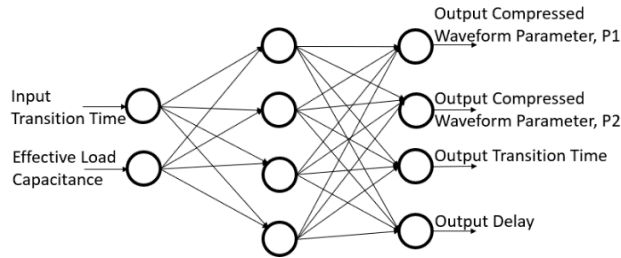


FIGURE 5.2 Combined NLDM-ECSM cell delay model

Table 5.5 Combined DL NLDM-ECSM Falling-edge NOR Model

Cell Type	NOR														
Model Type	Combined 1000 pts			Combined 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	8			6			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	1.27	2.16	27.98	1.08	1.57	10.21	0.7	1.3	34.7	0.39	0.66	8.7			
0.5 VDD%	0.84	1.41	14.84	1.00	1.36	8.93	0.5	0.8	8.4	0.29	0.54	9.37			
0.2 VDD%	1.32	1.97	21.48	1.02	1.39	8.19	0.7	1.1	34.9	0.49	0.85	12.0			
Mean corrccoef	0.99911			0.99603			0.9995			0.99914					
WF MSE	7.62E-05			2.36E-04			5.2E-05			5.10E-05					
Delay	0.59	0.82	3.86	0.52	0.78	6.09	Mean: 0.48 Std: 0.51 Max: 3.56			0.53	0.91	15.46			
Delay MSE	1.27E-05			9.53E-06			1.76E-05								
Transition	1.42	1.81	9.11	1.33	1.78	9.61	Mean: 1.19 Std: 2.06 Max: 8.00			1.65	2.09	33.49			
Transition MSE	8.34E-05			2.45E-05			1.76E-05								

Same ECSM Autoencoders models are used for NOR, INV, NAND waveforms. Separate models are better by ~2x for waveform values and by ~1.2x for delay and transition time. However, combined models are still better than NLDM-LUT in delay and transition time maximum percentage error. Combined rising-edge and falling-edge 150-points DL-NLDM outperformed

100x100 and lower sizes of NLDM-LUT tables in terms of the maximum percentage errors.

Table 5.6 Combined DL NLDM-ECSM Falling-edge INV Model

Cell Type	INV														
Model Type	Combined 1000 pts			Combined 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	8			6			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	1.67	3.21	20.12	1.52	2.17	16.96	0.7	1.3	34.7	0.39	0.66	8.7			
0.5 VDD%	1.06	2.42	38.82	1.29	1.85	15.17	0.5	0.8	8.4	0.29	0.54	9.37			
0.2 VDD%	1.43	3.04	33.84	1.28	1.86	15.03	0.7	1.1	34.9	0.49	0.85	12.0			
Mean corrcoeff	0.99875			0.98943			0.9995			0.99914					
WF MSE	9.35E-05			4.70E-04			5.2E-05			5.10E-05					
Delay	1.18	1.67	11.98	0.84	1.29	9.18	Mean: 0.60 Std: 0.94 Max: 10.41						0.29	0.71	33.1
Delay MSE	2.94E-05			1.66E-05			3.48E-05								
Transition	4.06	4.16	27.98	2.77	3.62	27.49	Mean: 1.91 Std: 3.0 Max: 22.67						0.62	3.76	59.8
Transition MSE	4.38E-04			1.04E-04			3.48E-05								

Table 5.7 Combined DL NLDM-ECSM Falling-edge NAND Model

Cell Type	NAND														
Model Type	Combined 1000 pts			Combined 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	8			6			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	1.65	3.68	28.65	1.06	1.43	9.26	0.7	1.3	34.7	0.39	0.66	8.7			
0.5 VDD%	1.00	1.59	33.96	0.98	1.29	8.11	0.5	0.8	8.4	0.29	0.54	9.37			
0.2 VDD%	1.19	1.87	32.37	0.98	1.31	8.26	0.7	1.1	34.9	0.49	0.85	12.0			
Mean corrcoeff	0.99909			0.99776			0.9995			0.99914					
WF MSE	8.50E-05			1.56E-04			5.2E-05			5.10E-05					
Delay	0.83	1.15	5.81	0.53	0.82	7.85	Mean: 0.49 Std: 0.79 Max: 7.84						0.24	1.28	28.6
Delay MSE	2.14E-05			4.29E-05			2.91E-05								
Transition	1.01	1.41	9.85	1.02	1.37	8.44	Mean: 0.85 Std: 1.21 Max: 7.51						0.46	2.11	50.8
Transition MSE	5.06E-05			2.39E-05			2.91E-05								

Table 5.8 Combined DL NLDM-ECSM Rising-edge NOR Model

Cell Type	NOR														
Model Type	Combined 1000 pts			Combined 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	8			6			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	2.04	4.19	45.96	0.99	1.37	9.16	0.7	1.4	30.3	0.38	0.7	11.97			
0.5 VDD%	1.63	3.91	36.76	1.00	1.40	9.30	0.5	0.8	30.4	0.27	0.4	7.08			
0.2 VDD%	2.03	4.31	43.71	1.02	1.42	9.68	0.7	1.5	31.4	0.37	0.7	5.44			
Mean corrcoeff	0.99788			0.99108			0.9994			0.998632					
WF MSE	1.42E-04			3.50E-04			5.80E-05			7.27E-05					
Delay	0.82	1.21	8.54	0.94	1.29	6.95	Mean: 0.71 Std: 0.93 Max: 5.95						0.91	1.21	16.54
Delay MSE	1.71E-05			1.47E-05			6.29E-05								
Transition	3.79	5.81	45.47	3.32	5.1	47.39	Mean: 3.2 Std: 4.72 Max: 36.33						4.24	5.44	43.85
Transition MSE	1.88E-04			1.19E-04			6.29E-05								

Table 5.9 Combined DL NLDM-ECSM Rising-edge INV Model

Cell Type	INV														
Model Type	Combined 1000 pts			Combined 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	8			6			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	2.15	3.89	54.07	1.23	1.85	14.64	0.7	1.4	30.3	0.38	0.7	11.97			
0.5 VDD%	1.57	2.82	48.44	1.23	1.84	13.89	0.5	0.8	30.4	0.27	0.4	7.08			
0.2 VDD%	2.89	4.74	43.60	1.25	1.89	15.24	0.7	1.5	31.4	0.37	0.7	5.44			
Mean corrcoeff	0.99783			0.98039			0.9994			0.998632					
WF MSE	1.80E-04			7.30E-04			5.80E-05			7.27E-05					
Delay	1.94	2.79	36.93	2.0	2.34	25.35	Mean: 0.41 Std: 2.67 Max: 42.25						2.82	3.72	51.53
Delay MSE	5.25E-05			5.73E-05			2.35E-04								
Transition	6.64	9.24	137.08	5.28	8.26	14.74	Mean: 4.86 Std: 4.86 Max: 51.80						9.25	12.13	128.6
Transition MSE	4.49E-04			4.00E-04			2.35E-04								

Table 5.10 Combined DL NLDM-ECSM Rising-edge NAND Model

Cell Type	NAND														
Model Type	Combined 1000 pts			Combined 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	8			6			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	2.04	3.65	24.37	1.01	1.42	10.68	0.7	1.4	30.3	0.38	0.7	11.97			
0.5 VDD%	1.63	2.35	19.75	1.00	1.39	10.35	0.5	0.8	30.4	0.27	0.4	7.08			
0.2 VDD%	2.76	3.59	39.25	0.99	1.36	8.79	0.7	1.5	31.4	0.37	0.7	5.44			
Mean corrcoeff	0.99805			0.98926			0.9994			0.998632					
WF MSE	1.70E-04			4.33E-04			5.80E-05			7.27E-05					
Delay	1.67	2.21	15.85	1.51	1.94	22.74	Mean: 0.79 Std: 1.34 Max: 12.30						1.88	2.58	27.38
Delay MSE	4.68E-05			3.97E-05			8.66E-05								
Transition	5.28	6.33	47.62	3.16	4.75	48.83	Mean: 2.69 Std: 4.56 Max: 74.66						5.54	6.13	57.95
Transition MSE	2.73E-04			1.68E-04			8.66E-05								

After training, the corresponding waveform decoder is used to decode the output waveform parameters, this decoded waveform is ready to be used in the ECSM model. Delay and transition time can still be used to perform the NLDM if needed.

5.4 DL-WFDM: Deep Learning Waveform-Delay Model

Deep learning waveform delay model, DL-WFDM, is proposed instead NLDM LUT or DL-NLDM models. In DL-WFDM, auto-encoded input/output waveforms parameters are proposed to model cell delays instead of the cell delay and the transition time used in the NLDM. Figure 5 shows our proposed new cell delay training methodology. First, each input waveform is encoded as well as its associated output waveform using the corresponding rising or falling edge Autoencoder encoder model. Since sigmoid activation function is used for the middle

Autoencoder layer i.e., the encoder output layer, there is no need to normalize the input waveform parameters or the expected output waveform parameters.

The encoded input waveform parameters and the associated effective capacitance values are applied as input data to train the deep learning network. The deep-learning network is trained on these encoded input waveform parameters in addition to the associated effective capacitance input value to produce the expected encoded output waveform parameters.

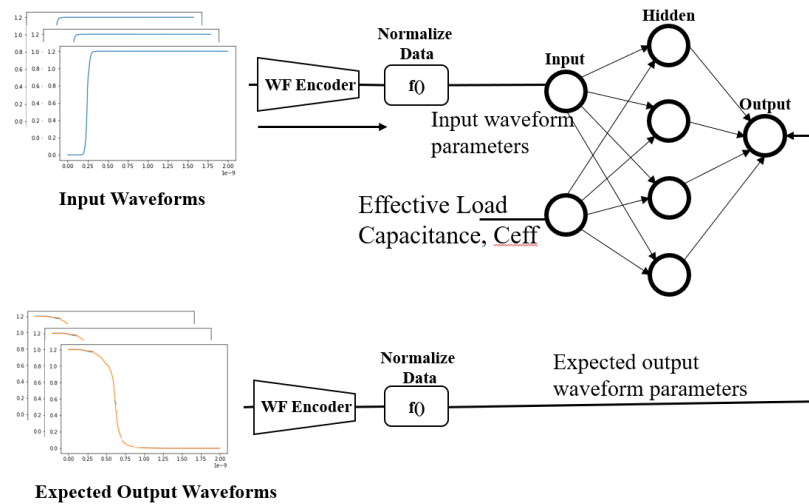


FIGURE 5.3 Training DL Cell model with encoded input/output waveform parameter Table 5.11 shows the proposed DL-WFDM model results. Using 1000-points sampled encoded waveforms to train the new model resulted in a standard deviation of percentage errors for key waveform points below 1.5% compared to SPICE simulation. These models also give below 5.75% delay percentage error compared to SPICE simulation. These DL-WFDM models, based on 1000-points waveforms, outperformed DL-WFDM that use 150-points sampled encoded waveforms by factors greater than 2x. Tables 5.10 to table 5.16 show also a comparison of combined DL-WFDM using 1000-pts encoded waveforms, DL-WFDM using 150-points encoded waveforms, separate DL-NLDM and 1000-pts and 150-pts encoded waveforms and 7x7 NLDM-LUT. The tables show that DL-WFDM using 1000-pts generally outperformed DL-WFDM in decoded waveform fidelity, and in average percentage delay error. However, the separate DL-NLDM and

1000-pts remain better than all the DL models, and better than 7x7 NLDM-LUT in maximum percentage error as mentioned before.

Table 5.11 DL-WFDM NOR Falling-edge Model

Cell Type	NOR														
Model Type	DL-WFDM 1000 pts			DL-WFDM 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	4			4			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	0.49	0.75	6.57	0.94	1.26	10.40	0.7	1.3	34.7	0.39	0.66	8.7			
0.5 VDD%	0.45	0.59	3.91	0.61	0.86	7.82	0.5	0.8	8.4	0.29	0.54	9.37			
0.2 VDD%	0.51	0.67	3.79	0.58	0.85	8.79	0.7	1.1	34.9	0.49	0.85	12.0			
Mean corrcoeff	0.99962			0.99752			0.9995			0.99914					
MSE	3.78E-5			1.43E-4			5.2E-05			5.10E-05					
Delay	2.27	2.37	11.07	3.3	3.93	17.2	Mean: 0.48 Std: 0.51 Max: 3.56						0.53	0.91	15.46

Table 5.12 DL-WFDM INV Falling-edge Model

Cell Type	INV														
Model Type	DL-WFDM 1000 pts			DL-WFDM 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	4			4			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	0.93	1.36	10.28	1.25	2.19	27.93	0.7	1.3	34.7	0.39	0.66	8.7			
0.5 VDD%	0.64	1.04	8.79	0.99	1.91	26.71	0.5	0.8	8.4	0.29	0.54	9.37			
0.2 VDD%	0.97	1.49	14.84	0.93	1.85	27.12	0.7	1.1	34.9	0.49	0.85	12.0			
Mean corrcoeff	0.99872			0.99235			0.9995			0.99914					
MSE	1.08E-4			3.55E-4			5.2E-05			5.10E-05					
Delay	4.22	5.75	32.3	5.95	8.28	59.43	Mean: 0.60 Std: 0.94 Max: 10.41						0.29	0.71	33.1

Table 5.13 DL-WFDM NAND Falling-edge Model

Cell Type	NAND														
Model Type	DL-WFDM 1000 pts			DL-WFDM 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	4			4			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	0.89	1.18	12.89	0.82	1.21	12.35	0.7	1.3	34.7	0.39	0.66	8.7			
0.5 VDD%	0.68	0.79	5.32	0.59	0.78	10.33	0.5	0.8	8.4	0.29	0.54	9.37			
0.2 VDD%	1.00	1.12	7.87	0.51	0.72	6.93	0.7	1.1	34.9	0.49	0.85	12.0			
Mean corrcoeff	0.99885			0.99804			0.9995			0.99914					
MSE	1.09E-4			1.39E-4			5.2E-05			5.10E-05					
Delay	3.04	2.93	21.47	2.57	2.61	13.19	Mean: 0.49 Std: 0.79 Max: 7.84						0.24	1.28	28.6

Table 5.14 DL-WFDM NOR Rising-edge Model

Cell Type	NOR														
Model Type	DL-WFDM 1000 pts			DL-WFDM 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	4			4			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	0.62	0.92	9.49	0.85	0.94	6.79	0.7	1.4	30.3	0.38	0.7	11.97			
0.5 VDD%	0.37	0.48	5.74	0.95	1.12	7.00	0.5	0.8	30.4	0.27	0.4	7.08			
0.2 VDD%	0.42	0.63	4.36	0.99	1.19	7.55	0.7	1.5	31.4	0.37	0.7	5.44			
Mean corrcoeff	0.99936			0.99536			0.9994			0.998632					
MSE	6.39E-5			1.74E-4			5.80E-05			7.27E-05					
Delay	2.83	2.88	14.53	6.02	4.92	25.59	Mean: 0.71 Std: 0.93 Max: 5.95						0.91	1.21	16.54

Table 5.15 DL-WFDM INV Rising-edge Model

Cell Type	INV														
Model Type	DL-WFDM 1000 pts			DL-WFDM 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	4			4			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	0.97	1.42	13.01	0.94	1.95	33.58	0.7	1.4	30.3	0.38	0.7	11.97			
0.5 VDD%	0.43	0.72	8.28	0.98	2.04	33.58	0.5	0.8	30.4	0.27	0.4	7.08			
0.2 VDD%	0.48	0.71	7.88	1.02	2.14	33.62	0.7	1.5	31.4	0.37	0.7	5.44			
Mean corrcoeff	0.9988			0.98901			0.9994			0.998632					
MSE	9.99E-5			3.6E-4			5.80E-05			7.27E-05					
Delay	2.83	5.51	14.53	8.59	12.49	90.75	Mean: 1.41 Std: 2.67 Max: 42.25						2.82	3.72	51.53

Table 5.16 DL-WFDM NAND Rising-edge Model

Cell Type	INV														
Model Type	DL-WFDM 1000 pts			DL-WFDM 150 pts			DL-NLDM & 1000pts Encoded ECSM			DL-NLDM & 150pts Encoded ECSM			NLDM-LUT 7x7		
Model ID	4			4			6 & 2			6 & 2					
	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max	Mean	Std	Max
0.8 VDD%	0.55	0.86	5.11	0.83	1.19	11.96	0.7	1.4	30.3	0.38	0.7	11.97			
0.5 VDD%	0.36	0.52	4.48	0.87	1.16	12.30	0.5	0.8	30.4	0.27	0.4	7.08			
0.2 VDD%	0.42	0.71	4.95	0.91	1.23	12.33	0.7	1.5	31.4	0.37	0.7	5.44			
Mean corrcoeff	0.99935			0.99387			0.9994			0.998632					
MSE	5.93E-5			2.3E-4			5.80E-05			7.27E-05					
Delay	3.00	3.35	22.79	7.60	8.23	35.49	Mean: 1.79 Std: 1.34 Max: 12.30						1.88	2.58	27.38

5.5.1 DL-WFDM: Multi-stage analysis

DL-WFDM represents a radical change to the exiting NLDM-LUT, or the DL variants proposed in this research, therefore it's important to evaluate the accumulation of model errors in multi-stages. Six stages of modeled cells, NOR INV, NAND, are stacked in a way that ensures signal transition at each stage like Fig. 5.4 for INV cells, similar setup is created for NOR and NAND wired to operate as inverters to switch signal at each stage. The first waveform is encoded using

the corresponding DL waveform Autoencoder to produce the input waveform parameters. The input parameter is applied to the corresponding cell DL-WFDM with the input capacitance to produce the output encoded waveform parameter. The output waveform parameters are then decoded to calculate the delay between the input and output waveforms. The output waveform parameters and the next stage input capacitance are applied to the second stage to produce the second stage output waveform parameters. The process is repeated for all stages, and the average percentage delay is calculated at each stage output.

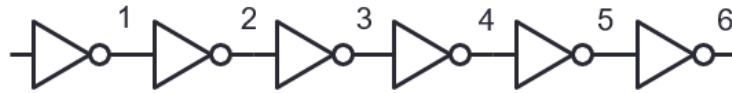


FIGURE 5.4 Multi-stage INV cell setup

Table 5.17 shows the results of this multi-stage experiment for each cell for different rising and falling starting edge. The results are not satisfactory compared with NLDM-LUT or DL-NLDM expected results but are promising to be enhanced with more training and powerful model.

Table 5.17 DL-WFDM Multi-stage analysis results

Cell Type	Multi-stage Analysis					
	NOR Avg% Delay Error		INV Avg% Delay Error		NAND Avg% Delay Error	
	Rising	Falling	Rising	Falling	Rising	Falling
Stage \ Input WF Edge						
1	2.6	3.2	5.1	4.0	4.1	2.9
2	6.1	5.7	9.6	6.3	10.6	3.4
3	6.3	4.9	7.1	7.4	4.2	8.9
4	4.9	6.2	6.5	6.7	7.8	3.6
5	6.2	5.3	6.6	6.3	3.6	6.9
6	5.4	6.1	6.3	6.8	6.9	3.3

5.5 Conclusions

DL-NLDM retrieving time outperformed this of 7x7 NLDM-LUT by 1.69x, and outperformed 100x100 NLDM-LUT by 12x using the outlined Python implementation. In addition, DL-NLDM outperforms 7x7 NLDM-LUT average percentage errors by ~1.24x better (delay time) and ~1.55x (transition time) for falling-edge waveforms. DL-NLDM has ~5x better delay time transition time average percentage errors for rising-edge waveforms. In addition, DL-NLDM outperforms

NLDM-LUT 100x100 maximum percentage errors by $\sim 2.37x$ better delay time and $\sim 2.x$ better transition time for falling-edge and by $\sim 3.19x$ better delay time and $\sim 2.75x$ better transition time for rising-edge waveforms. Compared to SPICE, DL-NLDM gives less than 0.60% average percentage delay and 1.91% average transition time percentage error, and it gives less than 1.41% average delay and 4.86% average percentage transition time error.

Combined DL-NLDM and ECSM encoded models showed consistent results for NOR, INV and NAND models. However, separate DL-NLDM and ECSM encoded waveform models remained better in average percentage errors by $\sim 2x$ for waveform values and by $\sim 2x$ for delay and transition time. Combined models are still better than NLDM-LUT delay and transition time maximum percentage error.

DL-WFDM shows promising results to propagate waveform parameters rather than transition/delay time. Separate DL-NLDM and ECSM encoded models remained better in average percentage errors by $\sim 2x$ for waveform values and by another $\sim 2x$ for delay and transition time. DL-WFDM models are still better than NLDM-LUT maximum delay time percentage error.

Chapter 6

S-RNN MOR: Structured Recurrent Neural Network Model Order Reduction for SISO, SIMO and MIMO LTI Systems

6.1 Overview

Many research efforts are still dedicated to finding optimal model order reduction techniques as in [Schilders'08]. MOR techniques fall into different categories; 1) Proper Orthogonal Decomposition (POD) methods that use a small set of uncorrelated coefficients that represented the whole system, 2) balanced truncation, BT, methods that reduce the state-space system while preserving the observable states, 3) reduced basis methods that preserve the most important basis of the original system and 4) ANN-based methods that can model the input/output relationship. Baziyad et al. [Baziyad'19] obtained 4th order model using BT technique, and used J-RNN [Jordan'97] model to estimate 4th order transfer function and used instrument variable, IV, method to estimate a third transfer function model. They also created a MOR framework that cascades the reduced-order systems obtained by the BT technique, and the J-RNN method to reduce the errors and uncertainties in both models. Salah et al. [Salah'16] proposed a simple RNN model structure model single input single output LTI system to a reduced 2nd order system, their work was limited to 2nd order SISO LTI system reduction. Nguyen et al. [Nguyen'19] used deep learning neural networks to extract Volterra kernels for I/O buffers from input/output data. In this section, we propose RNN model structure, S-RNN, to model SISO LTI systems to a reduced system of order N trained only on the original system step response. We also proposed an S-RNN network structure to model SIMO systems of O outputs with a reduced model of order N . The

main advantage of this S-RNN model is that the weights of the trained S-RNN model directly map to the discrete-time state-space parameters of the reduced dynamic system. Therefore, the discrete-time state-space reduced model can be directly retrieved from the trained model parameters. The training data are obtained by applying a step function at the input and discretize the step response at the output of the original systems. The training process applies step input to the input layer of the S-RNN and train the network to produce the expected step response at the S-RNN network output. Though simulation time is spent to produce the step response of the original white box system, the reduced-order system model can be used instead of the original model in subsequent simulations for faster simulation time. In addition, the same methodology can be used to model black box systems if their step responses are known or measured. The main contributions of this research paper are:

1. Proposed and trained an RNN network structure that models Nth order LTI SISO systems.
2. Proposed and trained an RNN network structure to model Nth order LTI SIMO and MIMO systems of any number of outputs O.
3. Modeled RLC interconnects with our proposed SIMO S-RNN model.
4. Reconstructed the continuous time state-space model from the trained S-RNN model.

6.2 Extracting Continuous-time Transfer Functions of RNN MOR Models

The data used to train the RNN is a discrete time data that is basically a sampled-time data of the full system model. Therefore, training the RNN model on this data results in a discrete time state space model of the original system. As mentioned before, the weights of the trained RNN represent the discrete space parameters. The weights of the trained RNN can be easily extracted to recover the discrete-time state-space model of the reduced system.

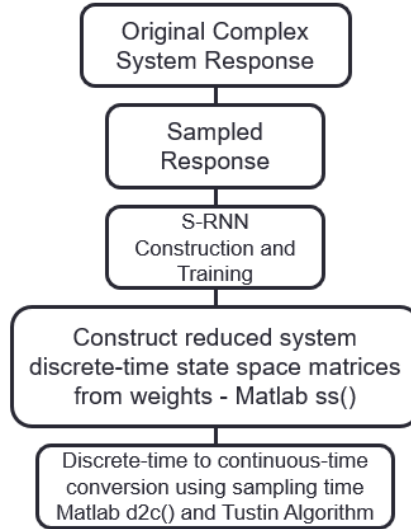


FIGURE 0-1 S-RNN Model Training and Continuous TF Extraction Flow

As shown in Fig. 6-1, we used Matlab function `ss()` to construct the reduced discrete-time state-space model using the trained RNN model parameters and the sampling time used to discretize the step response during training. Many mathematical algorithms exist to convert the discrete-time state-space model to its equivalent continuous-time model. Algorithms like zero-order hold, pole-zero match or Tustin algorithms [Beale'18] can be used to perform this task. We used Matlab function `d2c()` and Tustin algorithm to reconstruct the continuous-time model. Once the continuous-time state-space model is constructed, the continuous-time transfer function can be obtained. The transfer function of the trained RNN MOR systems can be obtained for any SISO, SIMO or MIMO systems of any order.

6.3 Model Order Reduction of SISO LTI Systems

6.3.1 Second order LTI SISO System

The general state space representation of second order LTI system that has a single input signal $U(t)$ and a single output signal $Y(t)$ is:

$$\begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} = \begin{bmatrix} a11 & a21 \\ a12 & a22 \end{bmatrix} * \begin{bmatrix} X1(t + 1) \\ X2(t + 1) \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \end{bmatrix} * [u1 \quad 0] \quad (1)$$

$$[Y(t)] = [c1 \ c2] * \begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} + [d1 \ d2] * \begin{bmatrix} U1(t) \\ 0 \end{bmatrix} \quad (2)$$

If we consider the case where the output signal is only a function of system states with no direct contribution of input signals, then the governing output signal equation is:

$$[Y(t)] = [c1 \ c2] * \begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} \quad (3)$$

Fig. 6-2 shows our implementation of the RNN network structure that models a SISO LTI system of a second order. The activation function is linear in all the layers. The delay of the feedback loop used is one delay element as explained in Matlab neural network toolbox user's guide.

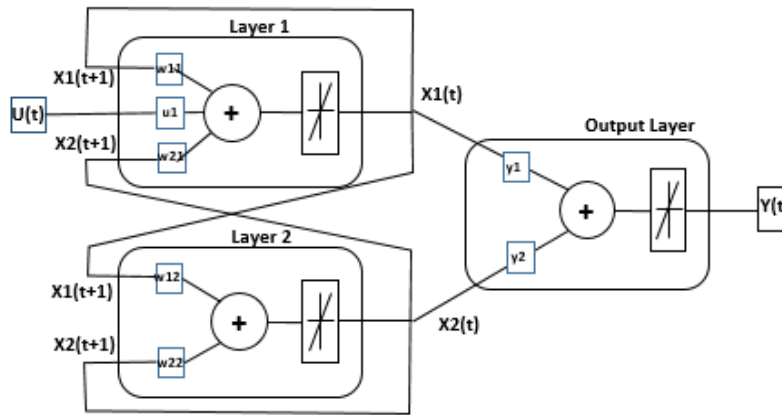


FIGURE 0-2 Second Order Discrete SISO System RNN Implementation

The Feedback loop from the output of layers 1&2 perceptron play as delay elements, hence the corresponding equations that govern the operation of this RNN are:

$$\begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} = \begin{bmatrix} w11 & w21 \\ w12 & w22 \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \end{bmatrix} + \begin{bmatrix} u1 \\ 0 \end{bmatrix} * [U(t) \ 0] \quad (4)$$

$$[Y(t)] = [y1 \ y2] * \begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} \quad (5)$$

Therefore, there is a direct mapping between this RNN implementation and the second order SISO LTI system. If we can train this RNN topology to represent a certain system, then the RNN weights can directly construct the discrete space second order model representing the trained system. This RNN implementation can be extended to third order model by introducing a third layer to the RNN connecting it to the other layers as shown in the following section.

6.3.2 Third order LTI SISO System

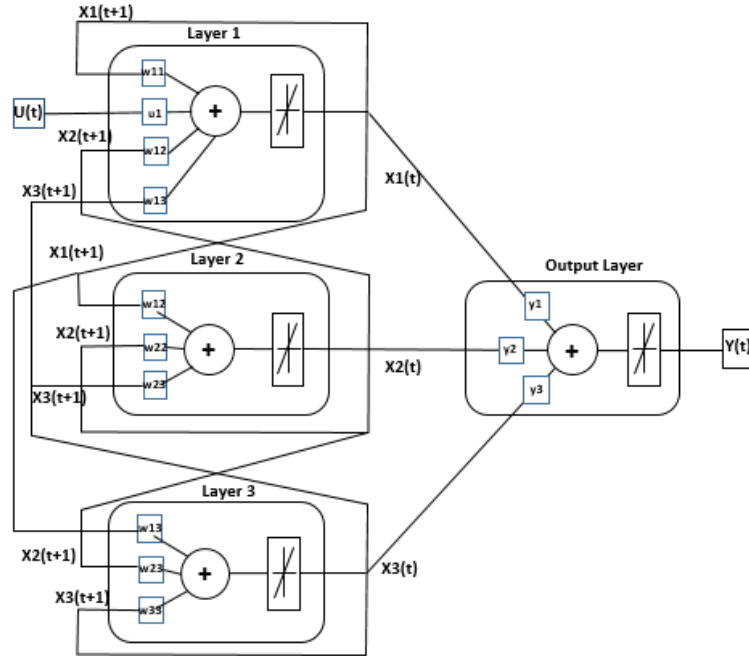


FIGURE 0-3 Third Order Discrete SISO System RNN Implementation

The state space representation of this third order model is:

$$\begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \end{bmatrix} = \begin{bmatrix} w11 & w12 & w13 \\ w21 & w22 & w23 \\ w31 & w32 & w33 \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \\ X3(t+1) \end{bmatrix} + \begin{bmatrix} u1 \\ 0 \\ 0 \end{bmatrix} * [U(t) \ 0 \ 0] \quad (6)$$

$$[Y(t)] = [y1 \ y2 \ y3] * \begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \end{bmatrix} \quad (7)$$

6.3.3 Nth Order LTI SISO System

We propose an RNN network structure in Fig. 6-4, that can model any SISO LTI system to a model of order N. The state space representation of this LTI system of order N and X of Nx1 state vector, U(t) single input and Y(t) single output is shown is equation (8) and (9). Thus, to model Nth order LTI system we need N+1 number of layers.

We used Matlab 2018a neural network toolbox 7 [Beale'18] on a Core i7, 8 GB physical memory to model our RNN implementation. Bayesian Regularization (BR) as the training algorithm, and the mean-squared-error as the loss function as well as maximum of 5000 epochs for all experiments. We were able to use this experiment setup to construct and train S-RNN model

order as large as 15. Convergence of S-RNN model order greater than 15 remains a challenging task that needs further research.

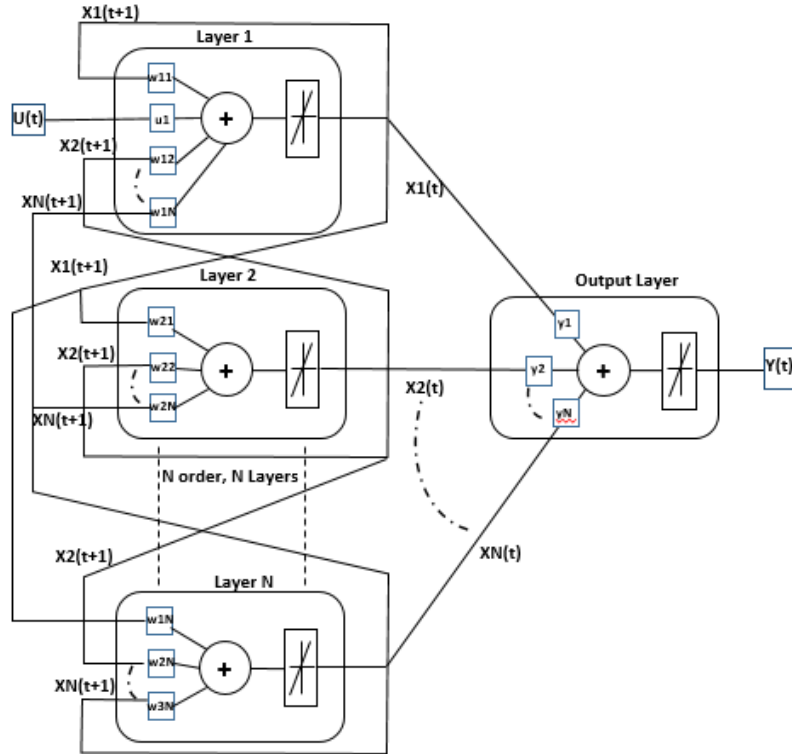


FIGURE 0-4 N Order Discrete SISO System RNN Implementation

The state space representation of this RNN implementation of the Nth order LTI system is:

$$\begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \\ \vdots \\ XN(t) \end{bmatrix} = \begin{bmatrix} w11 & w12 & w13 & \dots & w1N \\ w21 & w22 & w23 & \dots & w2N \\ w31 & w32 & w33 & \dots & w3N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ wN1 & wN2 & wN3 & \dots & wNN \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \\ X3(t+1) \\ \vdots \\ XN(t+1) \end{bmatrix} + \begin{bmatrix} u1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (8) * [U(t) \ 0 \ 0 \ \dots \ 0]$$

$$[Y(t)] = [y1 \ y2 \ y3 \ \dots \ yN] * \begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \\ \vdots \\ XN(t) \end{bmatrix} \quad (9)$$

6.4 SISO LTI RNN Modeling Experiments

6.4.1 Experiment 1

We used the same transfer function G in equation (10), used by Baziyad et al. [Baziyad'19] to be reduced to a 4th order system and compared the performance of our S-RNN MOR technique with the results of the MOR techniques used in that research. In addition to bode magnitude and phase

MSE metrics, we added the step response MSE as another metric for the comparison.

$$G - ORIG = \frac{s^4 - 20s^3 + 180s^2 - 840s + 1680}{s^6 + 22.5s^5 + 231s^4 + 1310s^3 + 3960s^2 + 5040s + 1680} \quad (10)$$

Training of our 4th order S-RNN converged at MSE of 8.76e-6 in 25 seconds, resulting in the transfer function in equation (11).

$$G - SRNN, 4 = \frac{0.002088s^4 - 0.2805s^3 + 2.293s^2 - 9.731s + 16.71}{s^4 + 10.51s^3 + 36.07s^2 + 49.23s + 16.7} \quad (11)$$

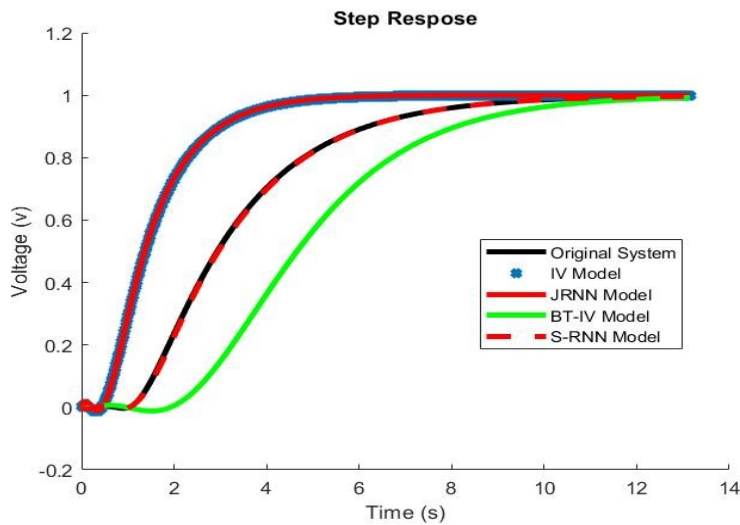


FIGURE 0-5 Step response of G-Orig, 4th Order S-RNN and Other Model Order Reduction Techniques

Fig. 6-5 shows the step response of G-Orig, 4th Order S-RNN and the other model order reduction techniques used by Baziyad et al. [Baziyad'19]. Table 6.1 shows the comparison of different approaches with our S-RNN 4th order model. S-RNN model gives better results than the proposed 4th order model BT-JRNN model proposed by Baziyad et al. In addition, S-RNN model outperformed IV and JRNN 4th order models used to evaluate the BT-JRNN performance. Training S-RNN on a 2nd order model converged in 19 seconds at 2.19e-4 MSE with transfer function shown in equation (12). The 2nd order model still gives better magnitude and step response results than IV, JRNN and BT-JRNN models, but worse phase MSE results than all of

them. Yang et al. model [Yang'96] remained the best in terms of magnitude MSE, but our proposed S-RNN 2nd and 4th order models are better when it comes to phase MSE.

Table 6.1 MSE Comparison of S-RNN Performance with Other MOR Techniques

MOR Technique	MSE Comparison		
	Magnitude	Phase	Step
IV [Baziyad'19]	0.0171	7.6253	0.414
IV-JRNN [Baziyad'19]	0.0182	7.7917	0.419
Yang et al. [Yang'96]	9.17e-7	95.59	NA
BT-JRNN [Baziyad'19]	0.0031	7.6862	0.414
S-RNN - 4 th order	5.5917e-5	6.4191	8.6105e-6
S-RNN - 2 nd order	2.3672e-4	14.2582	1.1777e-4

$$G - SRNN, 2 = \frac{0.002031s^2 - 0.2589s + 0.4322}{s^2 + 1.234s + 0.4371} \quad (12)$$

6.4.2 Experiment 2: Modeling Eady.mat Benchmark

We also tested our S-RNN proposed MOR technique with selected Matlab benchmark systems created by Chahlaoui et al. [Chahlaoui'02]. We used Eady benchmark that is defined by dense matrices with N=598 states. This benchmark models the atmospheric storm track.

6.4.2.1 S-RNN MOR of Order 5

We attempted to model this benchmark with order 5 reduced model. The RNN converged at MSE of 9.0e-5 resulting in the reduced TF given in equation (13). Fig. 6-6 and Fig. 6-7 show the step response and pole-zero diagrams of the full and 5th order reduced order systems respectively.

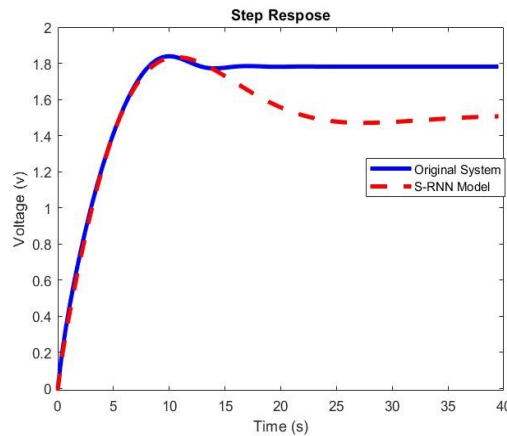


FIGURE 0-6 Step response of original “Eady” system and the 5th order MOR

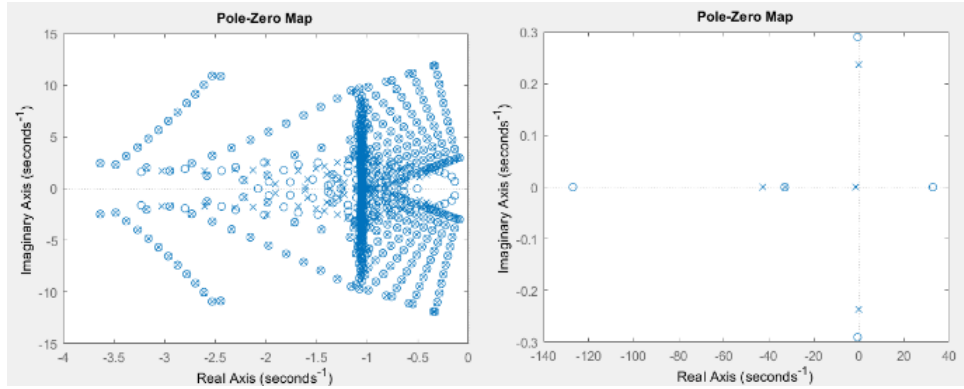


FIGURE 0-7 Pole-zero map of original and 5th order MOR

$$G - SRNN = \frac{0.02242s^5 - 0.5401es^4}{s^5 + 65.1s^4 + 1109s^3 + 1653s^2 + 561.9s + 128.6} \quad (13)$$

6.4.2.2 S-RNN MOR of Order 10

However, increasing the order of the reduced model resulted in better results. The new S-RNN model converged with $9.04e-6$ MSE in 1000 iterations and 27 sec resulting in the reduced system TF in equation (14). Table 6.2 compares the step response, magnitude, and phase MSE of the 5th and 10th S-RNN models. Fig. 6-8 shows the step response of the full and 10th order reduced order systems, and Fig. 6-9 shows the pole-zero plot of the original and the reduced model.

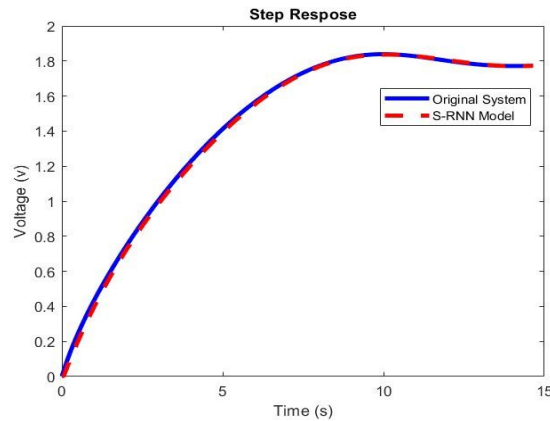


FIGURE 0-8 Step response of original Eady system and the 10th order

Table 6.2 MSE Comparison of 5th and 10th Order S-RNN Models

	S-RNN Model Order	
	5 th Order	10 th Order
Magnitude	3.25E-4	4.87E-4
Phase	34.75	23.64
Step	0.037	2.147E-4

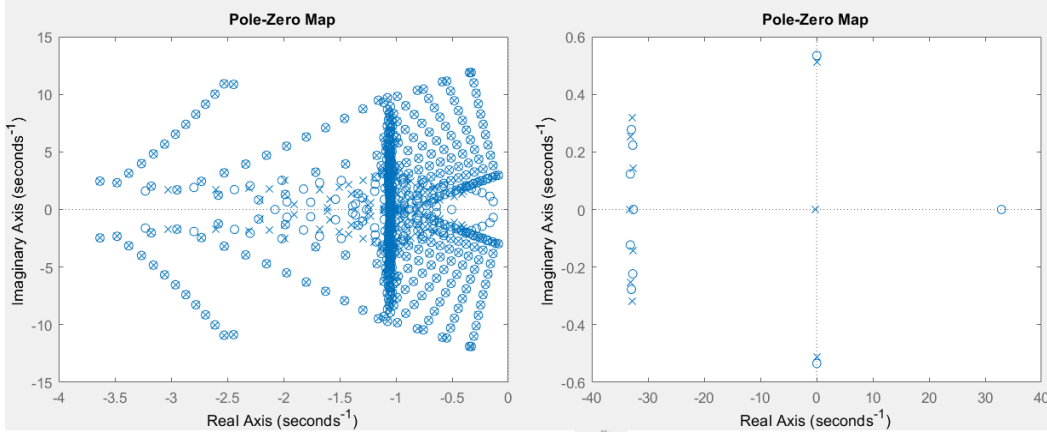


FIGURE 0-9 Pole-zero plot of original and reduced 10th order system

$$G - SRNN = \frac{-0.01405s^{10} - 2.771s^9 - 212.5s^8 - 6976s^7 + 656.8s^6 + 7.569e6s^5 + 2.483e8s^4 + 3.492e9s^3 + 1.905e10e10s^2 + 3.061e8s + 5.514e9}{s^{10} + 230.4s^9 + 2.275e4s^8 + 1.249e6s^7 + 4.118e7s^6 + 8.158e8s^5 + 9.02e9s^4 + 4.353e10s^3 + 1.045e10s^2 + 1.093e10s + 2.872e9} \quad (14)$$

6.4.3 Experiment 3: Modeling pde.mat Benchmark

6.4.3.1 S-RNN MOR of Order 3

The second benchmark we used is pde.mat with 84 number of states. Trying MOR to a system of order 3, the RNN training converged with 2.11e-7 MSE error rate.

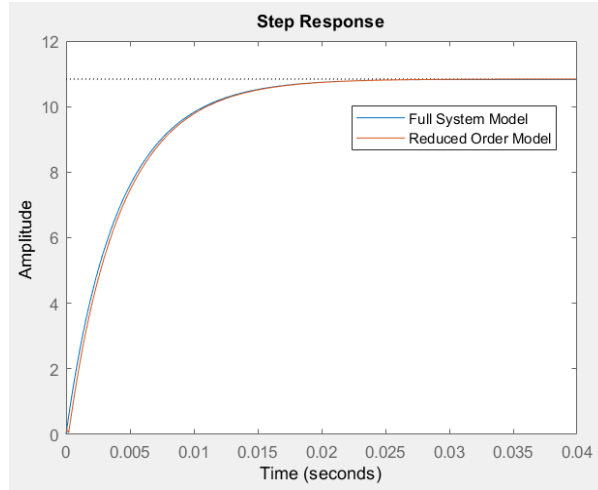


FIGURE 0-10 Step Response of the reduced 3rd order model

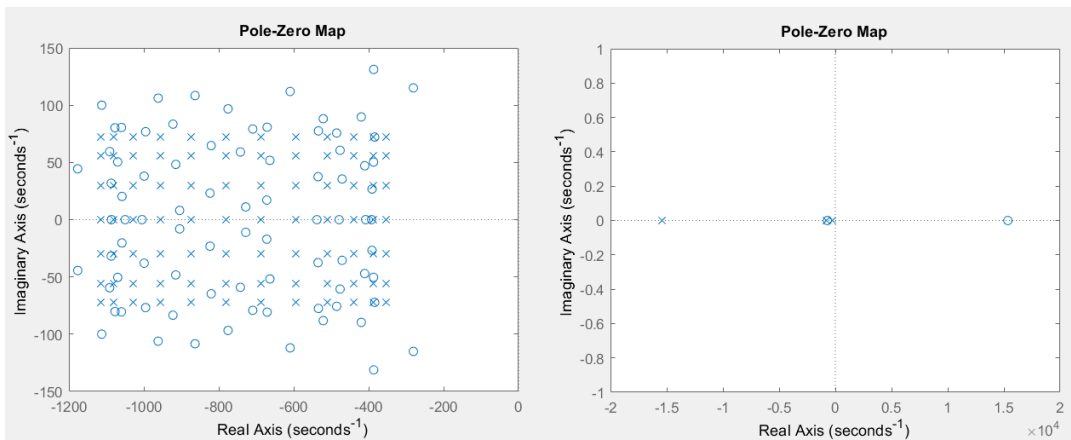


FIGURE 0-11 Pole-zero plot of the 3rd order model

$$G - SRNN = \frac{0.1844s^3 - 5530s^2 + 3.947e07s + 3.037e10}{s^3 + 1.642e04s^2 + 1.585e07s + 2.801e09} \quad (15)$$

6.5 Model Order Reduction of SIMO LTI Systems

SIMO systems are of particular importance in modeling interconnects between an output pin of a certain gate to the driven input pins of the subsequent gates. We extended the RNN state space modeling technique to model SIMO LTI systems of a single input and any number of O outputs and Nth order. Fig. 6-12 shows our proposed structured RNN network that achieves this SIMO model.

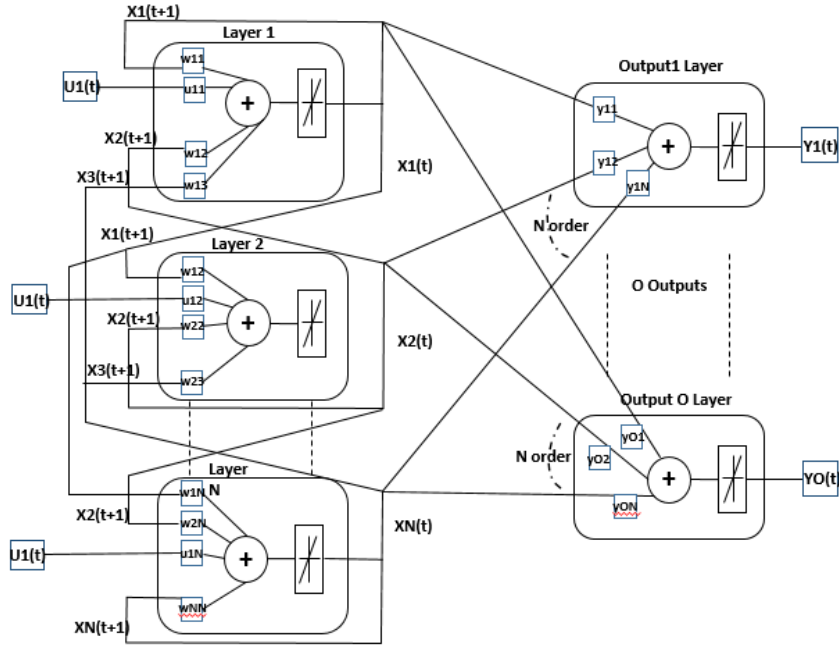


FIGURE 0-12 Nth Order, Single Input, O Outputs Discrete MIMO System Implementation

Equations 16 and 17 are the state space represented by this S-RNN given that:

- N is the order of the system.
- Single input signal.
- is the number of output signals.

$$\begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \\ \vdots \\ XN(t) \end{bmatrix} = \begin{bmatrix} w11 & w12 & w13 & \dots & w1N \\ w21 & w22 & w23 & \dots & w2N \\ w31 & w32 & w33 & \dots & w3N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ wN1 & wN2 & wN3 & \dots & wNN \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \\ X3(t+1) \\ \vdots \\ XN(t+1) \end{bmatrix} + \begin{bmatrix} u11 \\ u12 \\ u13 \\ \vdots \\ u1N \end{bmatrix} * U1(t) \quad (16)$$

$$\begin{bmatrix} Y1(t) \\ Y2(t) \\ \vdots \\ YO(t) \end{bmatrix} = \begin{bmatrix} y11 & y12 & y13 & \dots & y1N \\ y21 & y22 & y23 & \dots & y2N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y01 & y02 & y03 & \dots & y0N \end{bmatrix} * \begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \\ \vdots \\ XN(t) \end{bmatrix} \quad (17)$$

6.6 Experiment: Modeling RLC Interconnect SIMO LTI RNN

We selected arbitrary sub-circuit RLC interconnect as in the Fig. 6-13. The resistors value is $1.0e+03\Omega$, capacitances value is $1.76e-15F$, and inductances value is $0.245e-6H$. We connected 12 such sub-circuits together denoted as TRn in Fig. 6-14. The whole RLC interconnect represented 108 states. We then simulated this interconnect using a Spice simulator with a step response input

at v(1) node. The number of output pins is 4 at selected points, pins v(3), v(8), v(12) and v(14), in the RLC interconnect.

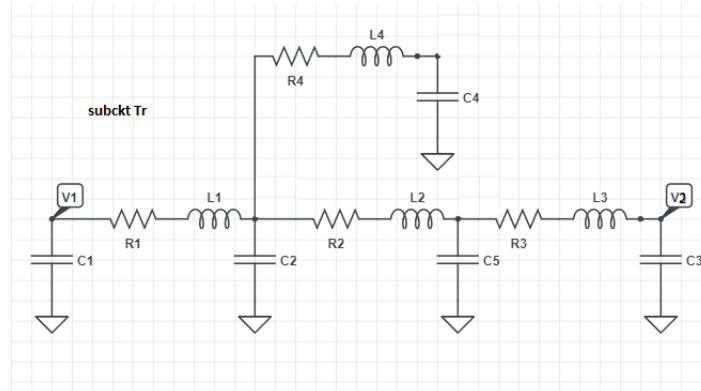


FIGURE 0-13 RLC Spice sub-circuit



FIGURE 0-14 RLC Transmission Line, v(3), v(8), v(12), v(14)

Single input multiple output RLC interconnect is a common connectivity structure in VLSI circuits. The single input v(1) represents the output of an active element. The multiple output pins v(3), v(8), v(12) and v(14) represent the input pins of the next-stage active elements fed by v(1). The RLC passive elements represent the wires connecting the active elements. We sampled the output waveforms at 2ps. Our 1x4 RNN SIMO network is trained on those multiple outputs using different RNN state space MOR order. RNN with order 2 converged with MSE of 5.1e-3 giving the following Fig. 6-15 and Fig. 6-16 results.

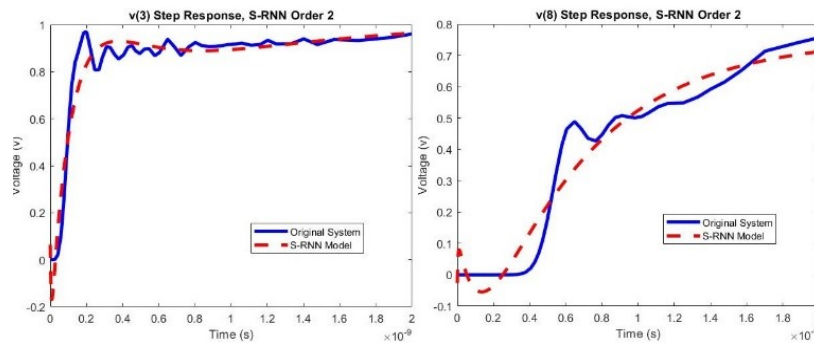


FIGURE 0-15 v(3), v(8) Spice and 2nd order SIMO MOR response

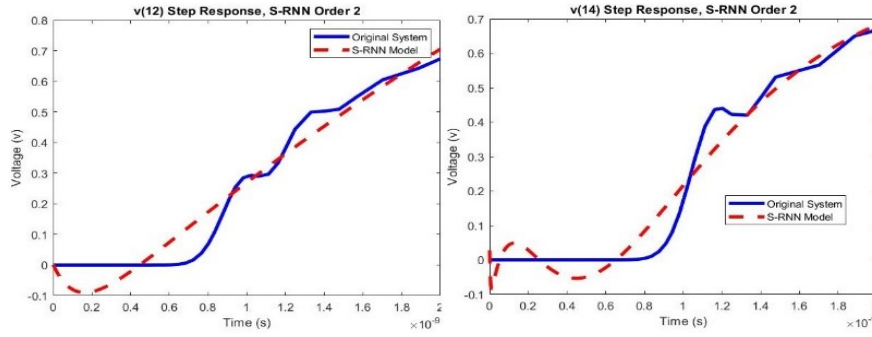


FIGURE 0-16 v(12), v(14) Spice and 2nd order SIMO MOR response

Training S-RNN of 5th order gave better results of MSE 9.1e-4 and the following Fig. 6-17 and Fig. 6-18 step response results.

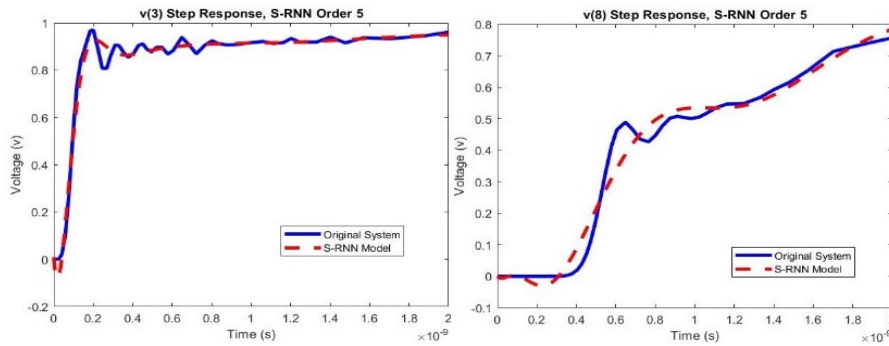


FIGURE 0-27 v(3), v(8) spice and 5th order SIMO MOR response

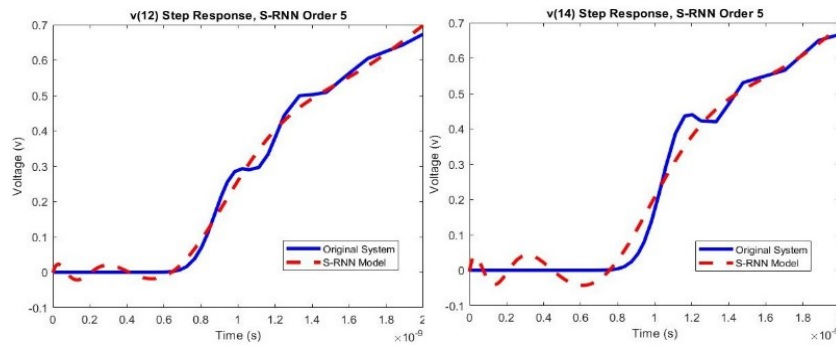


FIGURE 0-3 v(12), v(14) spice and 5th order SIMO MOR response

Modeling the RLC interconnect network using a single SIMO RNN network produces one network that models the whole system with all its outputs in one single training operation. However, the RNN weights are optimized to model all the outputs given a step input. Modeling each input/output as a single RNN model is expected to give better results, however it takes

training as many RNN networks as the number of outputs.

Modeling each input/output as a SISO 5th order Model gave better MSE results as in Fig. 6-19 and Fig. 6-20. Modeling output V(3) MSE is 2.0×10^{-4} , output V(8) MSE is 4.0×10^{-4} , output V(12) MSE is 3.0×10^{-4} , output V(14) MSE is 4.0×10^{-4} . Table 6.3 shows the MSE comparison of the three modeling experiments: 2nd order, 5th order SIMO and 5th order SISO models of the four output responses.

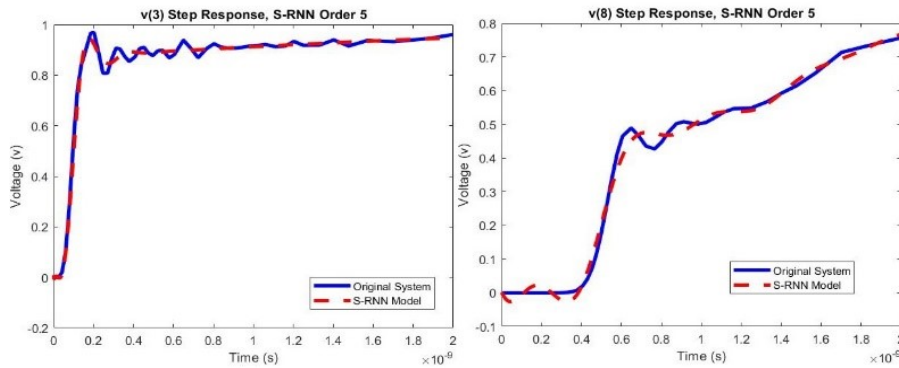


FIGURE 0-19 v(3), v(8) spice and 5th order SISO MOR response

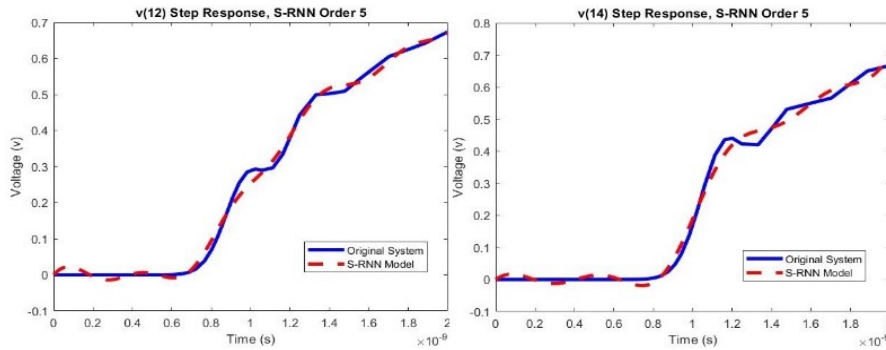


FIGURE 0-20 v(12), v(14) spice and 5th order SISO MOR response

Table 6.3 MSE Comparison of 3rd and 5th Order SIMO and 5th Order SISO S-RNN Models

Input	S-RNN Model Order					
	V(3)			V(8)		
	SIMO 2nd Order	SIMO 5th Order	SISO 5th Order	SIMO 2nd Order	SIMO 5th Order	SISO 5th Order
Step Response MSE	4.4E-3	5.53E-4	2.29E-4	6.9E-3	1.3E-3	9.55E-4
Input	V(12)			V(14)		
	SIMO 2nd Order	SIMO 5th Order	SISO 5th Order	SIMO 2nd Order	SIMO 5th Order	SISO 5th Order
Step Response MSE	3.0E-3	3.97E-4	2.03E-4	5.8E-3	1.1E-3	3.71E-4

6.7 Model Order Reduction of MIMO LTI Systems

6.7.1 Second order 2x2 LTI MIMO system

We extended the S-RNN state space modeling technique to model MIMO LTI systems of 2 inputs and 2 outputs second order system as illustrated in Fig. 6-21. The 2 inputs, 2 outputs second order LTI system is:

$$\begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} = \begin{bmatrix} a11 & a21 \\ a12 & a22 \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \end{bmatrix} + \begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix} * \begin{bmatrix} U1(t) \\ U2(t) \end{bmatrix} \quad (18)$$

$$\begin{bmatrix} Y1(t) \\ Y2(t) \end{bmatrix} = \begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix} * \begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} + \begin{bmatrix} d11 & d12 \\ d21 & d22 \end{bmatrix} * \begin{bmatrix} U1(t) \\ U2(t) \end{bmatrix} \quad (19)$$

Extending the idea of using RNN to represent 2 inputs, 2 outputs second order LTI system as follows:

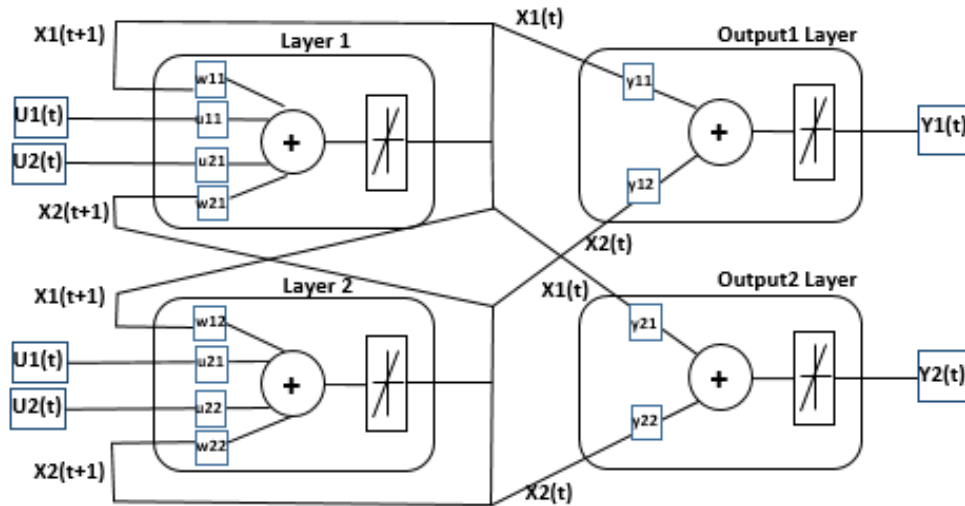


FIGURE 0-21 Second Order, 2 Inputs, 2 Outputs Discrete MIMO System RNN Implementation

The state space representation of such system is:

$$\begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} = \begin{bmatrix} w11 & w21 \\ w12 & w22 \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \end{bmatrix} + \begin{bmatrix} u11 & u21 \\ u21 & u22 \end{bmatrix} * \begin{bmatrix} U1(t) \\ U2(t) \end{bmatrix} \quad (20)$$

$$\begin{bmatrix} Y1(t) \\ Y2(t) \end{bmatrix} = \begin{bmatrix} y11 & y12 \\ y21 & y22 \end{bmatrix} * \begin{bmatrix} X1(t) \\ X2(t) \end{bmatrix} \quad (21)$$

6.7.2 Third order 2x2 LTI MIMO Systems

This 2x2 RNN state space modeling technique is generalized to model MIMO LTI systems of 3 inputs and 3 outputs, and 3 order as shown in Fig. 6-22.

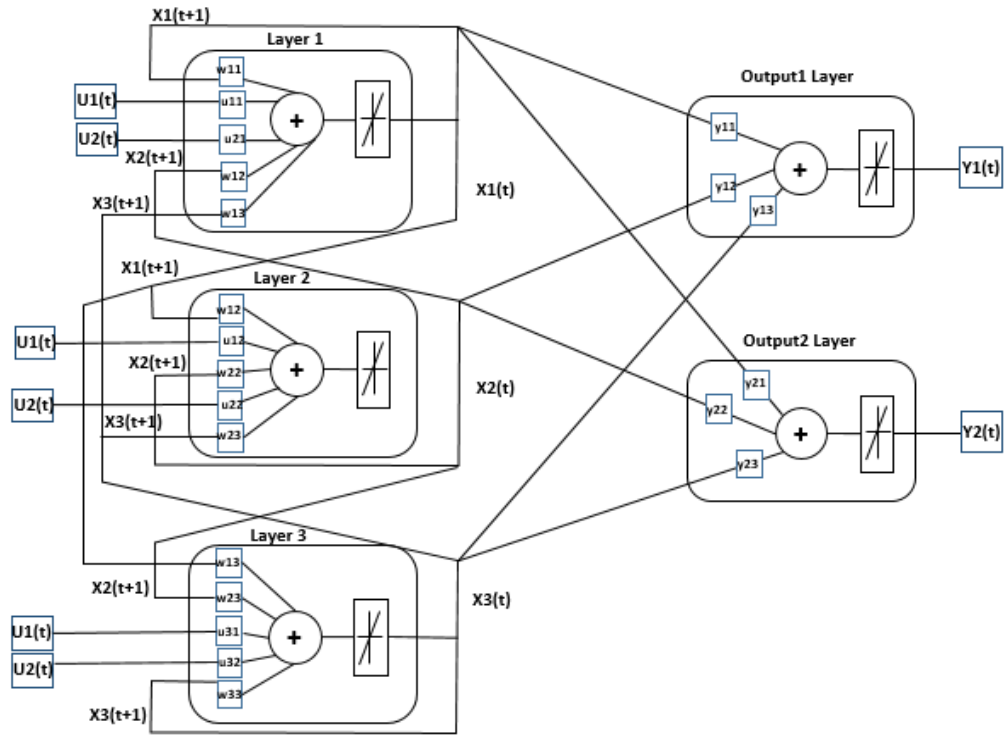


FIGURE 0-42 Third Order, 2 Inputs, 2 Outputs Discrete MIMO System RNN Implementation
The state space represented by this RNN is:

$$\begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \end{bmatrix} = \begin{bmatrix} w11 & w12 & w13 \\ w21 & w22 & w23 \\ w31 & w32 & w33 \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \\ X3(t+1) \end{bmatrix} + \begin{bmatrix} u11 & u21 \\ u12 & u22 \\ u13 & u23 \end{bmatrix} * \begin{bmatrix} U1(t) \\ U2(t) \end{bmatrix} \quad (22)$$

$$\begin{bmatrix} Y1(t) \\ Y2(t) \end{bmatrix} = \begin{bmatrix} y11 & y12 & y13 \\ y21 & y22 & y23 \end{bmatrix} * \begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \end{bmatrix} \quad (23)$$

6.7.3 Nth Order MxO LTI MIMO Systems

We further generalized this 3x3x3 RNN state space modeling technique to model MIMO LTI systems of any number M inputs and any number of O outputs and any system order N as illustrated in Fig. 6-23.

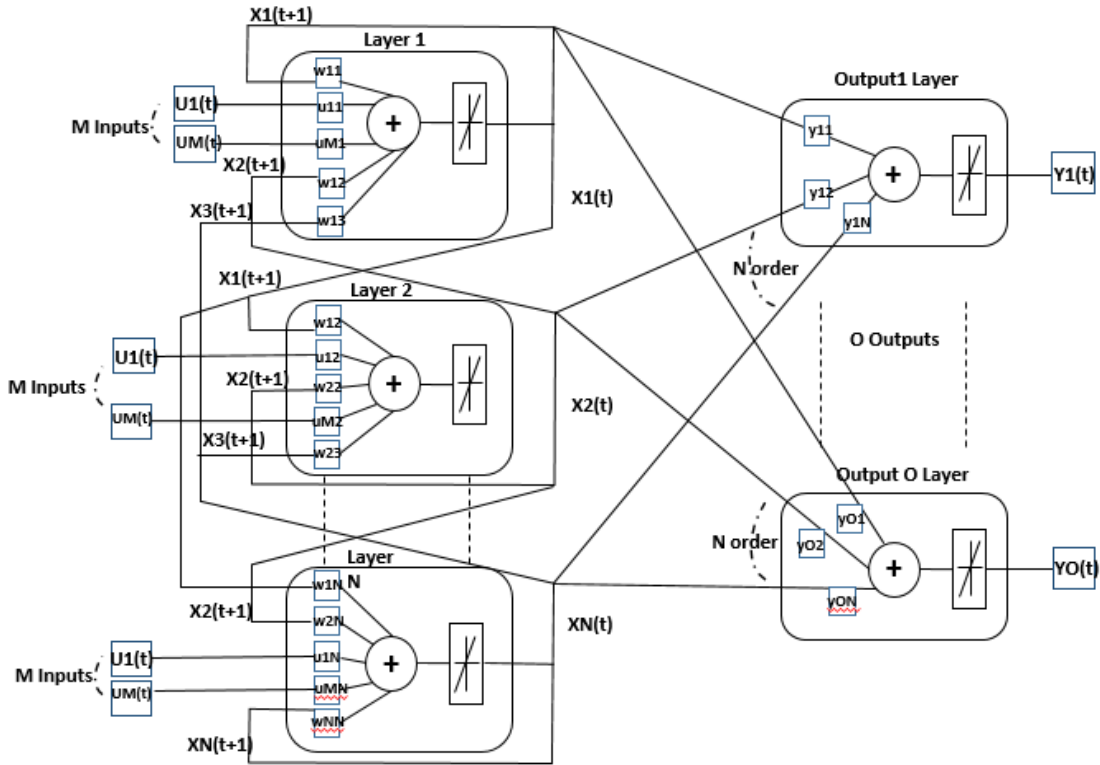


FIGURE 0-53 Nth Order, M Inputs, O Outputs Discrete MIMO System Implementation
Equations 24 and 25 are the state space represented by this S-RNN where:

- N is the order of the system.
- M is the number of input signals.
- is the number of output signals.

$$\begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \\ \vdots \\ XN(t) \end{bmatrix} = \begin{bmatrix} w11 & w12 & w13 & \dots & w1N \\ w21 & w22 & w23 & \dots & w2N \\ w31 & w32 & w33 & \dots & w3N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ wN1 & wN2 & wN3 & \dots & wNN \end{bmatrix} * \begin{bmatrix} X1(t+1) \\ X2(t+1) \\ X3(t+1) \\ \vdots \\ XN(t+1) \end{bmatrix} + \begin{bmatrix} u11 & u21 & \dots & uM1 \\ u12 & u22 & \dots & uM2 \\ u13 & u23 & \dots & uM3 \\ \vdots & \vdots & \ddots & \vdots \\ u1N & u2N & \dots & uMN \end{bmatrix} * \begin{bmatrix} U1(t) \\ U2(t) \\ U3(t) \\ \vdots \\ UM(t) \end{bmatrix} \quad (24)$$

$$\begin{bmatrix} Y1(t) \\ Y2(t) \\ \vdots \\ YO(t) \end{bmatrix} = \begin{bmatrix} y11 & y12 & y13 & \dots & y1N \\ y21 & y22 & y23 & \dots & y2N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ yO1 & yO2 & yO3 & \dots & yON \end{bmatrix} * \begin{bmatrix} X1(t) \\ X2(t) \\ X3(t) \\ \vdots \\ XN(t) \end{bmatrix} \quad (25)$$

6.7.4 Training of N*M*O MIMO S-RNN Models

The first attempt to train our proposed S-RNN MIMO followed the same approach we did to train SISO and SIMO S-RNN models, i.e., applying step response to the original system and use this step response to train the S-RNN model. The input data of the MIMO system is prepared by getting the step response at all outputs when a step function is applied on a single input having all other inputs wired to ground. This input is then wired to ground, and the step function is then applied to the next input to get the step response at all outputs for this input. If we apply the step function to different inputs of the S-RNN model in a sequence, the trained S-RNN will produce an output that is only correct if the input step function is applied in the same order used in training. To show this S-RNN MIMO training challenge, consider this example of the 7th order two-area power system described by the following state space matrices.

$$\begin{aligned} A &= \begin{bmatrix} -0.05 & 6 & 0 & -6 & 0 & 0 & 0 \\ 0 & -3.33 & 0 & 0 & 0 & 3.33 & 0 \\ 0 & 0 & -0.05 & 6 & 6 & 0 & 0 \\ 0.45 & 0 & -0.545 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3.33 & 0 & 3.33 & 0 \\ -5.21 & 0 & 0 & 0 & 0 & -12.5 & 0 \\ 0 & 0 & -0.521 & 0 & 0 & 0 & -12.5 \end{bmatrix}; \\ B &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 12.5 & 0 \\ 0 & 12.5 \end{bmatrix}; \\ C &= \begin{bmatrix} 0.178 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -0.6 & 0 & 0 \end{bmatrix}; \\ D &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}; \end{aligned}$$

Training the S-RNN MIMO model on the step function response of the first input followed by the step function response of the second input converged with 0.00116 MSE with 4th. order S-RNN model. Applying the step function on the generated MIMO transfer function resulted in the step response in Fig. 6-24 which clearly shows that there is a big deviation between the original and

the reduced systems. This deviation didn't improve by increasing the S-RNN model order.

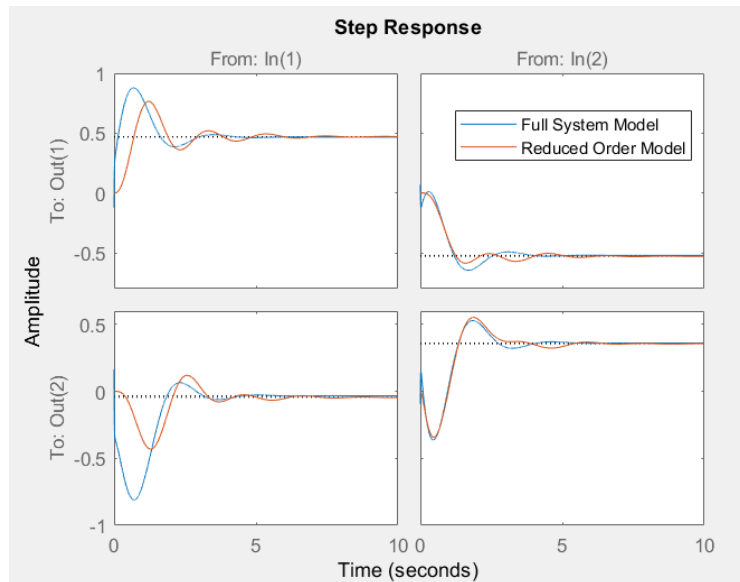


FIGURE 0-64 Step response of the original and fourth order reduced MIMO System

Training the proposed MIMO S-RNN requires a special input data preparation to avoid the problems of sequencing step responses in the training process. Instead of getting the step response of the original MIMO system, we get the MIMO system response to the sequence of input signals shown in Fig. 6-25 using Matlab lsim() function.



FIGURE 0-75 Sequence input to generate training MIMO data

This sequence guarantees that we train the S-RNN MIMO model on the response of individual input step functions while ensuring that the system is returned gracefully to its idle state during the period of which all input signals are set to ground. The input sequence also generates the MIMO response when all inputs are set to the step function. To make sure the input sequence is

not a factor of training the S-RNN model, the sequence is repeated with different combinations of which input is set to the step function and which inputs are set to ground.

6.8 S-RNN MIMO Model Order Reduction Experiments

6.8.1 Experiment 1: Jet Transport Aircraft example

To test our proposed S-RNN MIMO model order reduction, we used the Matlab MIMO State-Space Model of Jet Transport Aircraft example. The Jet example is 2x2 MIMO system. The inputs are the rudder and aileron, and the outputs are the yaw rate and the bank angle. There are four states defined for this model: beta, yaw, roll and phi.

$$\begin{aligned}
 A &= \begin{bmatrix} -0.0558 & -0.9968 & 0.0802 & 0.0415 \\ 0.5980 & -0.1150 & -0.0318 & 0 \\ -3.0500 & 0.3880 & -0.4650 & 0 \\ 0 & 0.0805 & 1.0000 & 0 \end{bmatrix}; \\
 B &= \begin{bmatrix} 0.0073 & 0 \\ -0.4750 & 0.0077 \\ 0.1530 & 0.1430 \\ 0 & 0 \end{bmatrix}; \\
 C &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \\
 D &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix};
 \end{aligned}$$

Matlab is used to generate the step response of this 2x2 MIMO system. Using the input sequence proposed in Fig. 6-25, we get the system response in Fig. 6-26 that will be used in training the S-RNN.

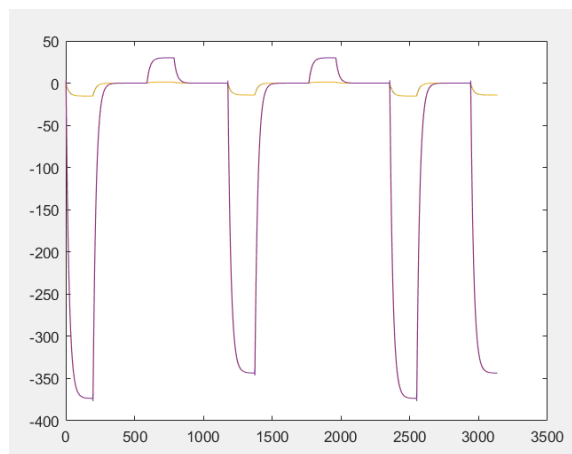


FIGURE 0-86 Response of the Original System to be Used in Training

6.8.1.1 Second Order S-RNN MIMO

Starting with RNN MIMO network of order 2, the training phase converged after 2553 iterations in 49 seconds with MSE step response of 0.00147. The step response of the full system is plotted against the reduced order model according to the following Fig. 6-27.

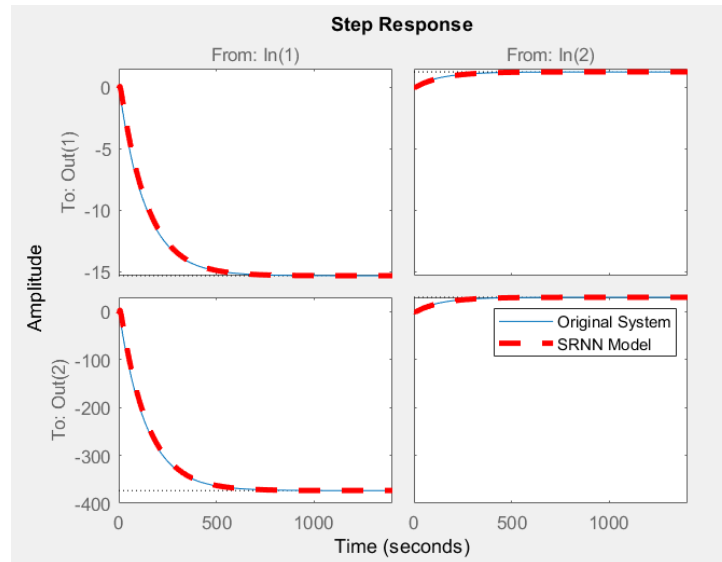


FIGURE 0-27 Step response of the original and second order reduced MIMO System

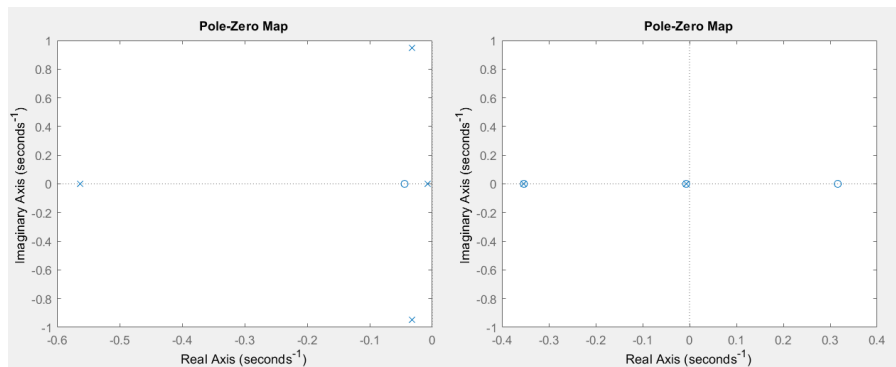


FIGURE 0-98 Pole-zero of the original and second order reduced MIMO System

Using the same technique used in our proposed SISO model, we obtained 4 different continuous time second order transfer functions. Those transfer functions describe the relationship between each output and each input of the MIMO system.

$$G - I1toO1 = \frac{-0.1436s^2 + 0.1202s - 0.0364}{s^2 + 0.2193s + 0.001542} \quad (26)$$

$$G - I1toO2 = \frac{-5.66s^2 + 3.611s - 0.576}{s^2 + 0.2193s + 0.001542} \quad (27)$$

$$G - I2toO1 = \frac{0.01166s^2 - 0.009704s + 0.001902}{s^2 + 0.2193s + 0.001542} \quad (28)$$

$$G - I2toO2 = \frac{0.4585s^2 - 0.2915s + 0.04634}{s^2 + 0.2193s + 0.001542} \quad (29)$$

6.8.1.2 Third Order S-RNN MIMO

To obtain a higher order MIMO models of potential complex systems, we used the same data to train a 3rd order S-RNN MIMO model order reduction network. The training phase converged after 5000 iterations, in 2:39 mins, with better MSE step response of 1.07E-4. The step response of the trained 3rd order S-RNN MIMO network is plotted against the full system response in the following figure.

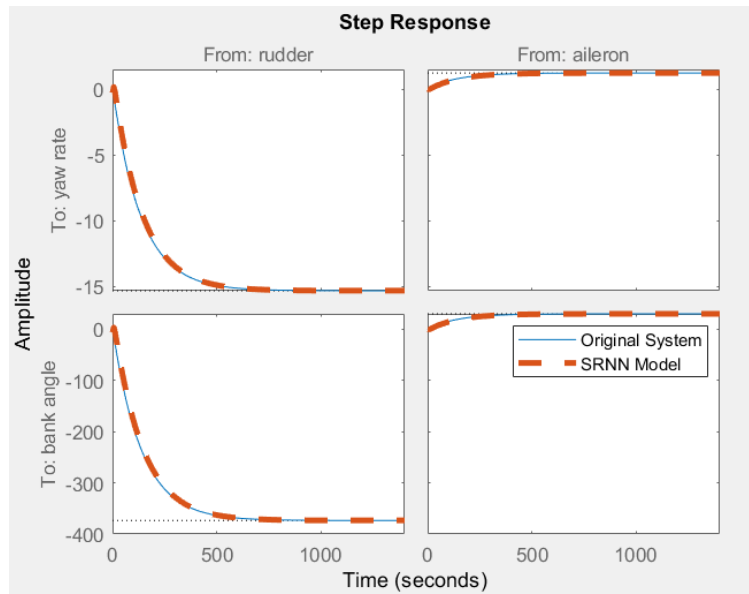


FIGURE 0-109 Step response of the original and third order reduced MIMO System

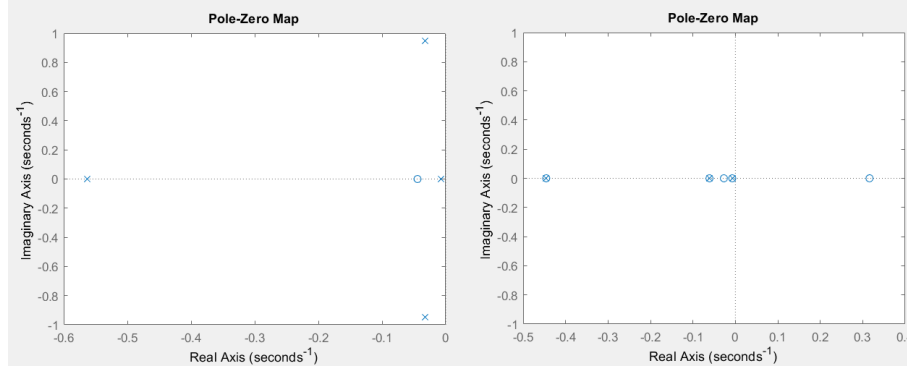


FIGURE 0-30 Pole-zero of the original and third order reduced MIMO System

The Matlab script we developed to extract the continuous-time transfer function generated the required 4 continuous time transfer functions of order 3. Table 6.4 shows MSE comparison of 2nd and 3rd order MIMO S-RNN models. The table shows that the 3rd order model didn't improve the accuracy of the reduced system.

$$G - I1toO1 = \frac{-0.3032s^3 + 0.1716s^2 - 0.02053s - 0.001076}{s^3 + 0.2852s^2 + 0.01167s + 7.022e^{-05}} \quad (30)$$

$$G - I1toO2 = \frac{-6.381s^3 + 3.744s^2 - 0.4628s - 0.02623}{s^3 + 0.2852s^2 + 0.01167s + 7.022e^{-05}} \quad (31)$$

$$G - I2toO1 = \frac{0.02664s^3 - 0.0143s^2 + 0.0015853s + 8.613e^{-05}}{s^3 + 0.2852s^2 + 0.01167s + 7.022e^{-05}} \quad (32)$$

$$G - I2toO2 = \frac{0.543s^3 - 0.3105s^2 + 0.03723s + 0.002111}{s^3 + 0.2852s^2 + 0.01167s + 7.022e^{-05}} \quad (33)$$

Table 6.4 MSE Comparison of MIMO 2nd and 3rd Order S-RNN

Input 1- Output 1	S-RNN Model Order		Input 1- Output 2	S-RNN Model Order	
	2 nd Order	3 rd Order		2 nd Order	3 rd Order
Magnitude	3.13	2.97	Magnitude	2.407	2.28
Phase	0.056	0.057	Phase	28.71	28.75
Step	0.1030	0.1229	Step	1.09E-4	6.3E-4
Input 2- Output 1	S-RNN Model Order		Input 2- Output 2	S-RNN Model Order	
	2 nd Order	3 rd Order		2 nd Order	3 rd Order
Magnitude	38.25	38.29	Magnitude	10.51	10.51
Phase	3.37E-4	4.55E-4	Phase	0.1901	0.1907
Step	30.64	39.04	Step	0.1983	0.2804

6.8.2 Experiment 2: Two-area power system example

6.8.2.1 Third Order S-RNN MIMO

The second selected example is the same system used to illustrate the training challenges presented in section 6.7.4. Using the input sequence proposed in Fig. 6-25, we get the system response in Fig. 6-31 that will be used in training the S-RNN.

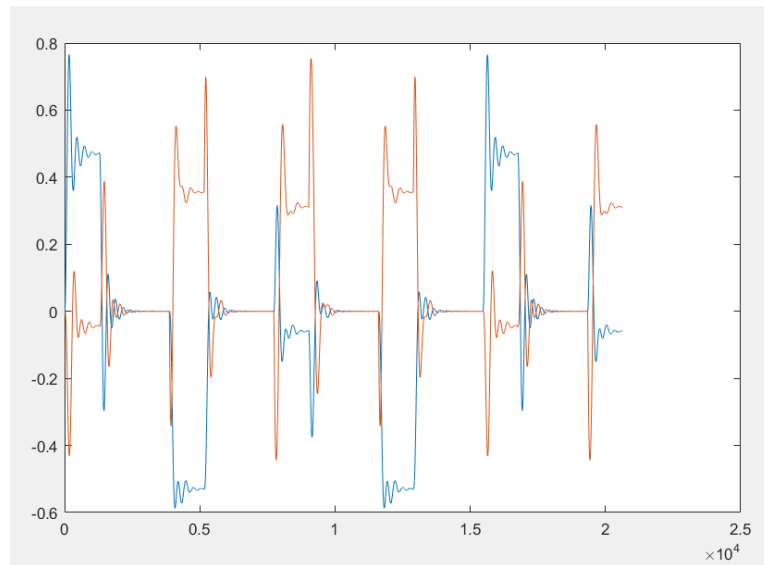


FIGURE 0-31 Response of the Original System to be Used in Training

The trained model gave better results at MSE step response of $1.36\text{E-}3$. This model converged in 7:46 mins, at 2327 iterations.

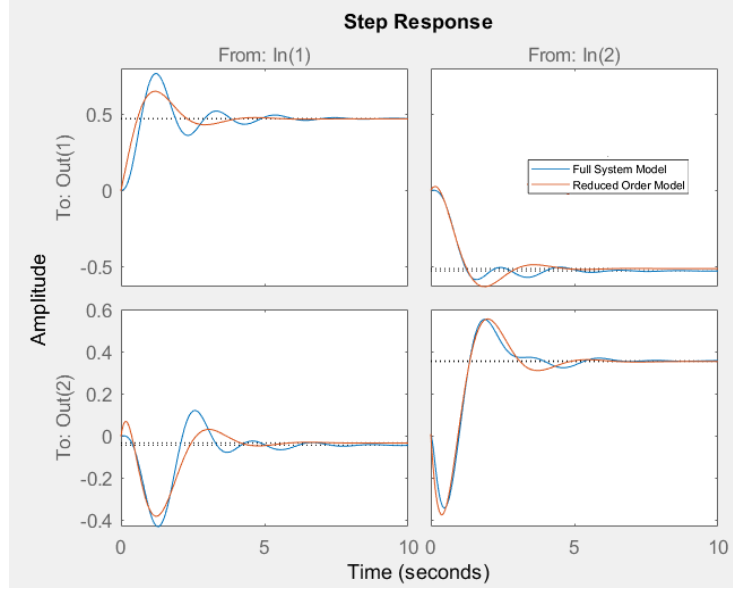


FIGURE 0-32 Step response of the original and 3rd order reduced MIMO System

$$G - I1to01 = \frac{-0.003258s^3 + 0.8751s^2 + 4.215s + 4.618}{s^3 + 4.185s^2 + 8.369s + 9.858} \quad (34)$$

$$G - I1to02 = \frac{-0.003413s^3 + 0.9459s^2 - 3.524s - 0.3328}{s^3 + 4.185s^2 + 8.369s + 9.858} \quad (35)$$

$$G - I2to01 = \frac{-0.001326s^3 + 0.3673s^2 - 1.286s - 5.075}{s^3 + 4.185s^2 + 8.369s + 9.858} \quad (36)$$

$$G - I2to02 = \frac{0.008092s^3 - 2.206s^2 - 1.516s + 3.473}{s^3 + 4.185s^2 + 8.369s + 9.858} \quad (37)$$

6.8.2.2 4th Order S-RNN MIMO

Modeling the same using a 4th order S-RNN gave better results with MSE step response of 4.17E-4 as table 6.5 clearly shows. This model converged in 9:54 mins, at 2031 iterations.

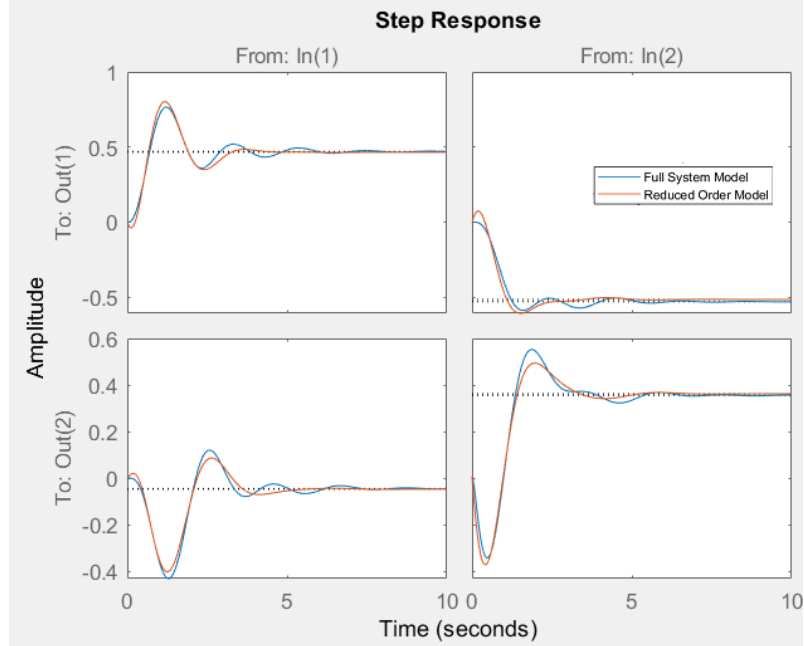


FIGURE 0-33 Step response of the original and 4th order reduced MIMO System

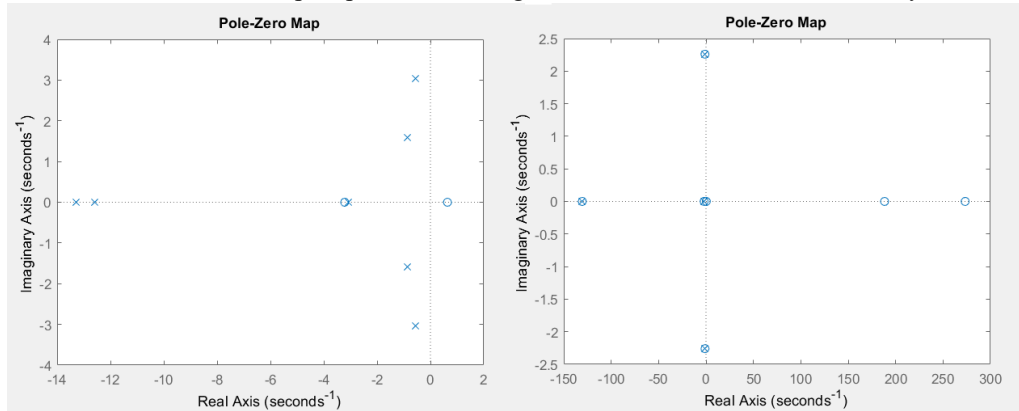


FIGURE 0-34 Pole-zero plot of the original and 3rd order reduced MIMO System

$$G - I1toO1 = \frac{0.002522s^4 - 0.7023s^3 + 3.492s^2 + 9.647s + 16.5}{s^4 + 3.804s^3 + 16.82s^2 + 24.2s + 35.48} \quad (38)$$

$$G - I1toO2 = \frac{-0.006871s^4 + 0.186s^3 + 0.5341s^2 - 8.929s - 1.663}{s^4 + 3.804s^3 + 16.82s^2 + 24.2s + 35.48} \quad (39)$$

$$G - I2toO1 = \frac{-0.003076s^4 + 0.8474s^3 - 1.806s^2 - 1.283s - 18.27}{s^4 + 3.804s^3 + 16.82s^2 + 24.2s + 35.48} \quad (40)$$

$$G - I2toO2 = \frac{0.006893s^4 - 1.875s^3 - 2.433s^2 - 10.22s + 12.85}{s^4 + 3.804s^3 + 16.82s^2 + 24.2s + 35.48} \quad (41)$$

Table 6.5 MSE Comparison of S-RNN of 3rd and 4th Models Performance

Input 1- Output 1	S-RNN Model Order		Input 1- Output 2	S-RNN Model Order	
	3 rd Order	4 th Order		3 rd Order	4 th Order
Magnitude	0.0327	0.0074	Magnitude	0.0171	0.0035
Phase	12.67	12.41	Phase	5.71	5.6916
Step	0.0025	6.2089e-04	Step	0.0027	4.5344e-04
Input 2- Output 1	S-RNN Model Order		Input 2- Output 2	S-RNN Model Order	
	3 rd Order	4 th Order		3 rd Order	4 th Order
Magnitude	0.0080	0.0085	Magnitude	0.0065	0.0028
Phase	4.58	4.577	Phase	4.922	4.9267
Step	0.0016	6.9822e-04	Step	5.8632e-04	4.9368e-04

6.8.3 Modeling MIMO LTI system using multiple 4th order S-RNN SISO Models

The SISO S-RNN MOR is used to generate reduced order models for the same MIMO system used in experiment 2 to compare the SISO performance with the MIMO S-RNN model order reduction. This approach requires M*O number of S-RNN models and training processes. Table 6.6 shows that while the 4 SISO S-RNN models gave better MSE step response, MIMO S-RNN model gives better magnitude and phase MSE.

6.8.3.1 Input 1 to Output 1 S-RNN SISO Model

Training a SISO 4th order S-RNN on the step response of input 1 to output 1 gave a MSE step response of 5.288e-04.

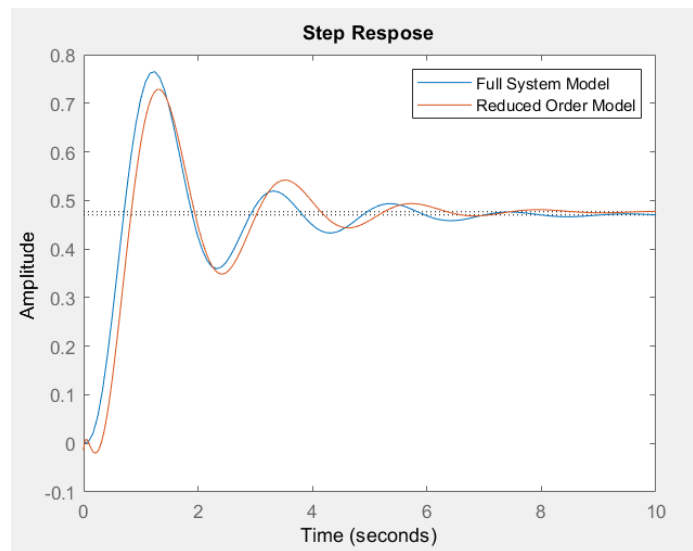


FIGURE 0-35 Step response of the original and fourth order input 1 to output 1 reduced MIMO System

$$G - I1/O1 = \frac{-0.0147s^4 + 0.5256s^3 + 5.219s^2 - 326.5s + 2439}{s^4 + 50.4s^3 + 673s^2 + 1157s + 5117} \quad (42)$$

6.8.3.2 Input 1 to Output 2 S-RNN SISO Model

Training a SISO 4th order S-RNN on the step response of input 1 to output 2 gave a MSE step response of 4.4828e-05.

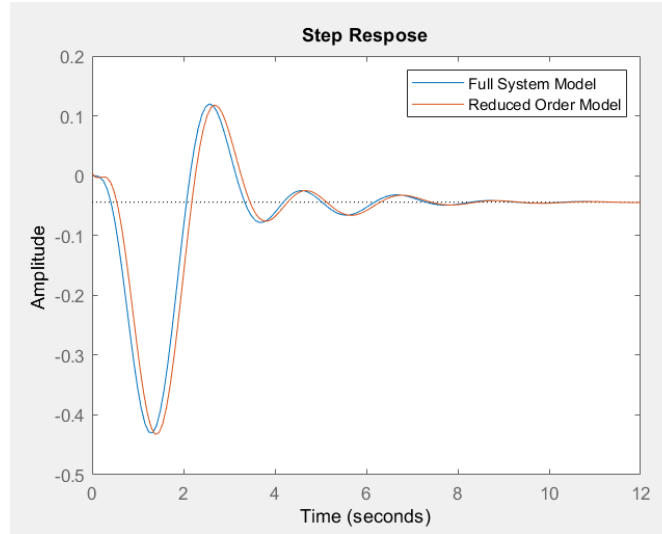


FIGURE 0-36 Step response of the original and fourth order input 1 to output 2 reduced MIMO System

$$G - I1/O2 = \frac{0.003778s^4 - 0.1421s^3 + 1.573s^2 - 8.273s - 1.345}{s^4 + 2.842s^3 + 14.68s^2 + 19.96s + 30.27} \quad (43)$$

6.8.3.3 Input 2 to Output 1 S-RNN SISO Model

Training a SISO 4th order S-RNN on the step response of input 2 to output 1 gave a MSE step response of 5.856e-04.

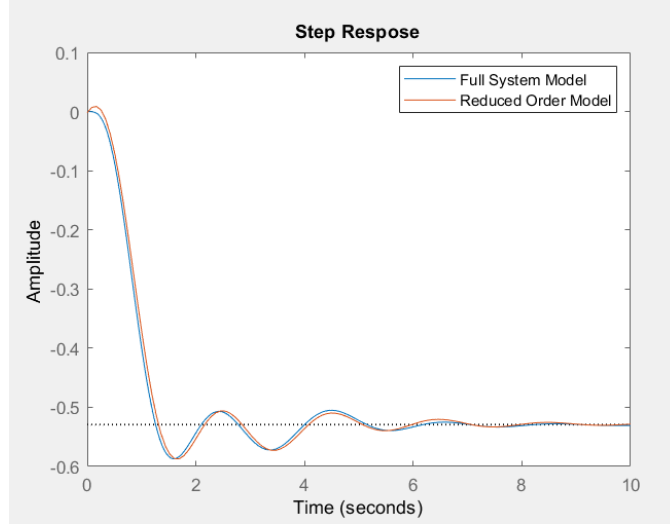


FIGURE 0-37 Step response of the original and fourth order input 2 to output 1 reduced MIMO System

$$G - I2/O1 = \frac{-0.001894s^4 + 0.1314s^3 - 0.3114s^2 - 3.446s - 12.67}{s^4 + 3.12s^3 + 14.58s^2 + 24.34s + 23.98} \quad (44)$$

6.8.3.4 Input 2 to Output 2 S-RNN SISO Model

Training a SISO 4th order S-RNN on the step response of input 2 to output 2 gave a MSE step response of 4.5314e-04.

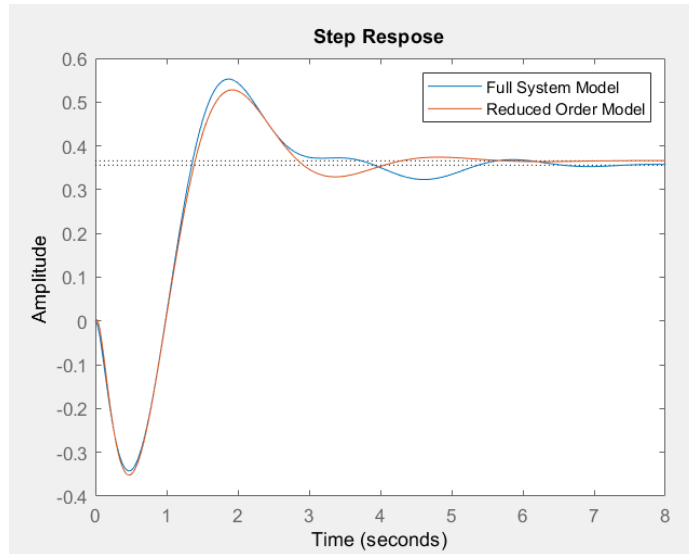


FIGURE 0-38 Step response of the original and fourth order input 2 to output 2 reduced MIMO System

$$G - I2/O2 = \frac{-0.001429s^4 - 0.05972s^3 + 157.6s^2 - 9470s + 9478}{s^4 + 344.9s^3 + 5199s^2 + 1.121e04s + 2.59e04} \quad (45)$$

Table 6.6 MSE Comparison of S-RNN of 3rd and 4th Models

Input 1- Output 1	S-RNN Model Order		Input 1- Output 2	S-RNN Model Order	
	4 th O SISO	4 th O MIMO		4 th O SISO	4 th O MIMO
Magnitude	0.002	0.0074	Magnitude	0.0228	0.0035
Phase	123.05	12.41	Phase	43.06	5.69
Step	5.288e-04	6.2089e-04	Step	4.4828e-05	4.5344e-04
Input 2- Output 1	S-RNN Model Order		Input 2- Output 2	S-RNN Model Order	
	4 th O SISO	4 th O MIMO		4 th O SISO	4 th O MIMO
Magnitude	0.0232	0.0085	Magnitude	0.0025	0.0028
Phase	26.61	4.577	Phase	35.58	4.926
Step	5.856e-04	6.9822e-04	Step	4.5314e-04	4.9368e-04

6.9 S-RNN Model Order Reduction Applications

We identified two main applications for our proposed S-RNN model order reduction technique:

- Modeling a black-box LTI system. Using the step response of the black-box system, the RNN can produce state space discrete model that models the system to a certain percentage error. If the percentage error is acceptable, then the state space model can be used to generate the corresponding continuous time transfer function of the black box system.
- Producing less computationally intensive model of a white-box complex LTI system. Using the step response of the complex system, the RNN can remodel the full system to lower order discrete time state space model. The corresponding continuous time state space models the full system according to a calculated percentage error.

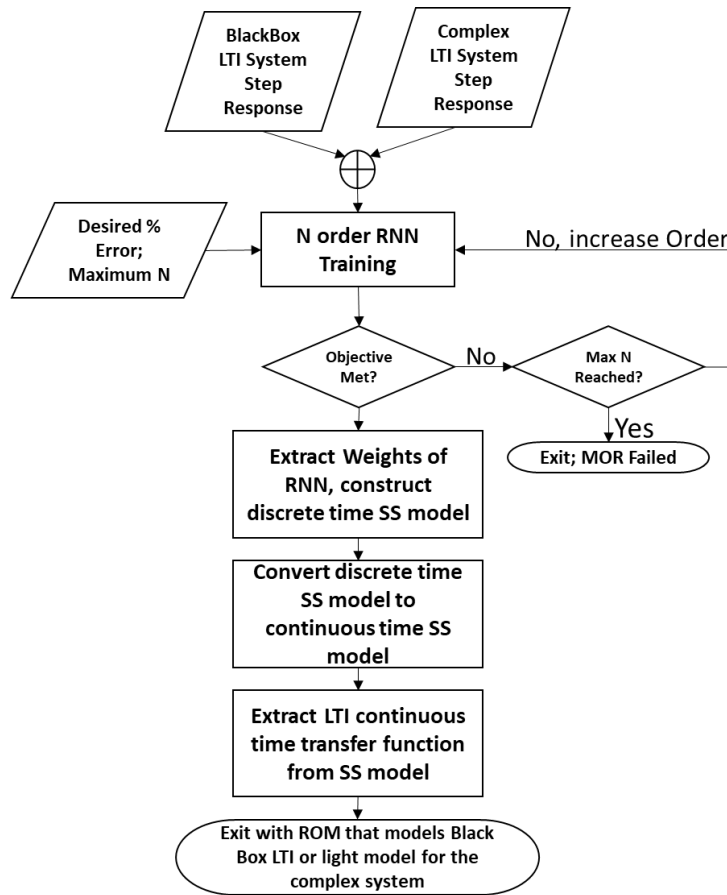


FIGURE 0-39 Using RNN model order reduction in black box identification

The proposed S-RNN model technique can be used in an algorithm to find the S-RNN model order that gives a desired mean-squared-error between the original and the reduced system response. The flow chart in Fig. 6-39 summarizes the proposed algorithm:

- The algorithm is fed with the step response of either the black box or the known complex LTI system. A desired percentage error is given as an input to the algorithm.
- The algorithm starts with $N=2$ as an initial value. The RNN of $N=2$ is trained, and the percentage error is calculated.
- If the percentage error is less than or equal the desired error, then the weights of the RNN are extracted. The weights directly map to the discrete time state space model matrices as it was explained in the previous sections.

- The discrete time state space model is converted to the continuous time and the transfer function is constructed from this model.

If the RNN percentage error is greater than the desired error, then the current model order is not enough to generate an acceptable reduced order model. The order N is incremented and the RNN is re-trained. The process continues until we reach an acceptable reduced model of order N , or we exceed the maximum allowed order given by clients of the algorithm.

Chapter 7

Conclusion and Future Work

In this research, DL-NLDM model is proposed, designed, and trained on input and output waveform delay and transition time in addition to capacitive load values. Two cell-delay models are trained on rising and falling edge waveform transitions. The proposed DL-NLDM generally outperformed the standard 7x7 NLDM-LUT in mean, standard deviation and maximum percentage errors compared to SPICE simulation. In addition, the proposed DL-NLDM outperformed the non-standard 100x100 LUT in maximum percentage errors. Waveform compression allows storing more complex waveforms data to raise the accuracy of the current source models without increasing the technology file size or degrading the performance. In this research, deep learning non-linear Autoencoders are used to compress voltage-time waveforms used in effective current source model. The performance of several deep learning Autoencoder models at different number of encoding parameters is evaluated against the SVD compression technique. Compression ratio of 104 at below ~1.5% percentage error standard deviation compared to SPICE simulation is achieved after encoding 1000-points sampled voltage-time waveforms. These compression ratios are 1.67x better than the nearest rank SVD results and ~39 to ~45x better than gzip and bz2 compression techniques. Better compression ratios are also achieved at less accuracy figures with other Autoencoder models and number of parameters. Autoencoding 150-points varying-time sampled waveforms with 2 parameters gives 1.79x better compression ratio than the nearest rank SVD compression with below 0.85% standard deviation of percentage error compared to SPICE simulation. Those Autoencoders gave ~40 to ~55x better than gzip and bz2 compression techniques. Autoencoders require large time to be trained, and

larger compression and decompression time compared to the SVD algorithm, however training and compression are offline operations that can be accepted for the sake of better compression and accuracy. The decompression time can be enhanced with better hardware resources and smaller encoding models. Building on both DL-NLDM and waveform compression, a combined DL-NLDM ECSM waveform model is proposed to produce both delay/transition time information as well as the compressed waveform parameters. Experiments show that separate DL-NLDM and Autoencoder ECSM waveform parameters are better than the combined ones. In addition, DL-WFDM is proposed to radically change transition/delay propagation to a full waveform propagation that can be used to measure the delay or perform ECSM delay calculations. Experiments show that separate DL-NLDM and Autoencoder ECSM waveform parameters still perform better than the proposed DL-WFDM models.

In addition, we proposed a structured RNN network that models SISO LTI system of any order N . A complex benchmark system of 598 states was reduced to a system of 10 states at $9.04e-6$ mean square error rate, using the proposed S-RNN SISO network. SISO 4th order S-RNN outperformed the reported MOR techniques using the same original transfer function. We also proposed an S-RNN network that can be trained to model any number of outputs, O , and order, N , of a SIMO system. This SIMO S-RNN network is used to model an RLC interconnect of 108 states in a reduced system of order 5 at $9.1e-4$ mean square error rate in a single training operation. Training individual interconnect outputs using SIMO S-RNN models gives better results but requires several training operations equals to the number of system outputs. The trained S-RNN weights are directly mapped to the discrete-time state-space model of the reduced system. Using the step-response sampling time and the trained network weights, the discrete-time state-space model is obtained and used to derive the continuous-time transfer function of the reduced model. Scaling S-RNN above 15th order system requires further research in RNN training algorithms.

Developing an S-RNN network to model a MIMO system of any number of inputs M , outputs O and order N is possible in theory. Training this MIMO S-RNN network is successful using the system response to a sequence of inputs that ensure the system is gracefully returned to the idle state after applying step input to each MIMO input.

Future work will focus on enhancing both training time and run time of the proposed DL-NLDM and DL-WFDM models which is key to adopt these models in practical digital design flows. Future work also includes enhancements to Autoencoders models and training algorithms to achieve higher compression ratios in less offline training time and less decoding time with lower error rates. In addition, some modern neural network could perform lossless compression, like bits-back coding, these models need to be examined and compared to the performance of the Autoencoder based compression. Developing new RNN training techniques is also recommended to guarantee S-RNN convergence for model order greater than 15.

References

- [**Abrishami'19**] M. S. Abrishami, M. Pedram and S. Nazarian, "CSM-NN: Current Source Model Based Logic Circuit Simulation - A Neural Network Approach" 2019 IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, 2019, pp. 393-400.
- [**Baziyad'19**] M. Baziyad, A. Jarndal and M. Bettayeb, "A Model Order Reduction Technique Based on Balanced Truncation Method and Artificial Neural Networks," 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), 2019, pp. 1-5, doi: 10.1109/ICMSAO.2019.8880270.
- [**Beale'18**] M. H. Beale, M. T. Hagan, H. B. Demuth, Neural Network Toolbox™ 7 User's Guide, 2018, , Matlab 2018rb release.
- [**Berz'15**] D. Berz, M. Engstler, M. Heindl, F. Waibel, "Comparison of lossless data compression methods", Technical Reports in Computing Science No. CS-07-2015, University of Applied Sciences Kempten.
- [**Capodiecì'17**] L. Capodiecì, "Data Analytics and Machine Learning for Design-Process-Yield Optimization in Electronic Design Automation and IC semiconductor manufacturing", CSTIC'17.
- [**Chahlaoui'02**] Y. Chahlaoui, P. Van Dooren, A collection of benchmark examples for model reduction of linear time invariant dynamical systems, SLICOT Working Note 2002-2, 2002.
- [**Chang'17**] W. Chang, C. Lin, S. Mu, L. Chen, C. Tsai, Y. Chiu, M. C.-T. Chao, "Generating Routing-Driven Power Distribution Networks With Machine-Learning Technique", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, 2017.

- [Dai'17] Y. Dai, R. K. Brayton, "Circuit recognition with deep learning", IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2017 IEEE'17.
- [Das'17] S. Das, J. R. Doppa, P. P. Pande, K. Chakrabarty, "Design-Space Exploration and Optimization of an Energy-Efficient and Reliable 3-D Small-World Network-on-Chip", "IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems", 2017.
- [Drmanac'09] D. G. Drmanac, F. Liu, Li-C. Wang, "Predicting variability in nanoscale lithography processes", Design Automation Conference (DAC), 2009.
- [Eremenko'17] K. Eremenko, H. de Ponteves, "Deep Learning A-Z™: Hands-On Artificial Neural Networks", Udemy Course: www.udemy.com
- [Fournier'19] Q. Fournier and D. Aloise, "Empirical Comparison between Autoencoders and Traditional Dimensionality Reduction Methods," 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2019, pp. 211-214, doi: 10.1109/AIKE.2019.00044.
- [Goodfellow'16] I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning". MIT Press, <http://www.deeplearningbook.org>, 2016, pp. 164-223 and pp. 499-523.
- [Hatami'09] S. Hatami, P. Feldmann, S. Abbaspour and M. Pedram, "Efficient compression and handling of current source model library waveforms," 2009 Design, Automation & Test in Europe Conference & Exhibition, 2009, pp. 1178-1183, doi: 10.1109/DATE.2009.5090841.
- [Hu'16] J. Hu, T. Li, S. Li, "Equivalence Checking between SLM and RTL Using Machine Learning Techniques", Quality Electronic Design (ISQED), 2016.
- [Jap'16] D. Jap, S. Bhasin W. He, "Supervised and unsupervised machine learning for side-channel based Trojan detection", IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)'16.

- [Jordan'97]** M. I. Jordan, *Serial Order: A Parallel Distributed Processing Approach*, *Neural-Network Models of Cognition - Biobehavioral Foundations Advances in Psychology*, pp. 471-495, 1997.
- [Kahng'11]** A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, "VLSI Physical Design: From Graph Partitioning to Timing Closure, Springer Netherlands, 2011", pp. 221-264.
- [Kahng'15]** A. B. Kahng, M. Luo, S. Nath, "SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects", "System Level Interconnect Prediction (SLIP) Workshop", 2015.
- [Lee'15]** W. Lee, Y. Kim, J. H. Ryoo, D. Sunwoo, A. Gerstlauer, L. K. John, "PowerTrain: A Learning-based Calibration of McPAT Power Models", *International Symposium on Low Power Electronics and Design (ISLPED)*, 2015.
- [Li'21]** L. Li and W. Yu, "Efficient and Accuracy-Ensured Waveform Compression for Transient Circuit Simulation," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1437-1449, July 2021, doi: 10.1109/TCAD.2020.3020496.
- [Mandouh'16]** E. El Mandouh A. G. Wassal "Accelerating the Debugging of FV Traces Using K-Means Clustering Techniques", *11th International Design & Test Symposium (IDT)*, 2016.
- [Mitchell'97]** T. Mitchell, *Machine Learning*, McGraw-Hill, New York, ISBN: 0070428077, 1997.
- [Nguyen'19]** T. Nguyen, X. Wang, X. Chen and J. Schutt-Aine, "A Deep Learning Approach for Volterra Kernel Extraction for Time Domain Simulation of Weakly Nonlinear Circuits," *2019 IEEE 69th Electronic Components and Technology Conference (ECTC)*, 2019, pp. 1889-1896, doi: 10.1109/ECTC.2019.00291.
- [Park'16]** S. J. Park, H. Yu, M. Swaminathan, "Preliminary application of machine-learning techniques for thermal-electrical parameter optimization in 3-D IC", *IEEE International Symposium on Electromagnetic Compatibility (EMC)*, 2016.

- [Provan'12]** P. Gregory, F. Alexander, "Machine-learning-based circuit synthesis", IEEE 27th Convention of Electrical & Electronics Engineers in Israel (IEEEI), 2012.
- [Ramalingam'07]** A. Ramalingam, A. K. Singh, S. R. Nassif, M. Orshansky and D. Z. Pan, "Accurate Waveform Modeling using Singular Value Decomposition with Applications to Timing Analysis," 2007 44th ACM/IEEE Design Automation Conference, 2007, pp. 148-153.
- [Salah'16]** K. Salah, A. Adel, Model order reduction using artificial neural networks, IEEE International Conference on Electronics, Circuits and Systems (ICECS), Monte Carlo, 2016, pp. 89-92.
- [Saurabh'18]** S. Saurabh and P. Mittal, "A Practical Methodology to Compress Technology Libraries Using Recursive Polynomial Representation," 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), 2018, pp. 301-306, doi: 10.1109/VLSID.2018.80.
- [Schilders'08]** W. H. Schilders, H. A. van der Vorst, J. Rommes, Model order reduction: theory, research aspects and applications, The European Consortium for Mathematics in Industry, Springer, ISBN 978-3-540-78841-6, 2008.
- [Sharma'16]** R. Sharma, "Characterization and Modeling of Digital Circuits", ISBN-10: 1983144827, paripath.inc, 2016, pp.75-120.
- [Si2'21]** Silicon Integration Initiative, <https://si2.org/os-downloads>
- [Synopsys'21]** Synopsys Inc., CCS Timing Technical White Paper version 2.0
- [Tenace'17]** V. Tenace, A. Calimera, "Activation-Kernel Extraction Through Machine Learning", New Generation of CAS (NGCAS) ,2017
- [Wang'15]** Li-C. Wang "Data Mining in Functional Test Content Optimization ", ASP-DAC, 2015.

- [Wang'17]** Li-C. Wang, "Experience of Data Analytics in EDA and Test – Principles, Promises, and Challenges", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, 2017.
- [Williams'95]** J. R. Williams, D. Zipser, "Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity", Back-propagation Book, L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1995, pp. 433-486.
- [Wu'17]** H. Wu, "Improving SAT-solving with Machine Learning", SIGCSE '17, ACM library.
- [Yang'96]** Z.-J. Yang, T. Tsuji, and T. Hachino, "Model reduction with time delay combining the least-squares method with the genetic algorithm", IEE Proceedings - Control Theory and Applications, vol. 143, no. 3, pp. 247-254, Jan. 1996.
- [Yang'17]** H. Yang, J. Su, Y. Zou, B. Yu, E. F. Y. Young, "Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning", "Design Automation Conference (DAC)", 2017.
- [Zhang'18]** S. Zhang, J. Zhai, J. Chen, Q. HE, "Autoencoder and its various variants", IEEE International Conference on Systems, Man, and Cybernetics, 2018.
- [Zheng'15]** X. Zheng, P. Ravikumar, L. K. John, A. Gerstlauer, "Learning-based Analytical Cross-Platform Performance Prediction", 15th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2015.

APPENDIX A

Artificial Intelligence, Machine Learning and Deep Learning Background

Machine learning, neural networks and deep learning are all concerned with producing a program that can learn from experience. Those programs usual tackle problems that don't scale linearly with increased complexity level or data size. Such problems are usually named NP-Complete problems. Machine or deep learning approach usually trades accuracy with performance. In other words, those ML/DP programs perform certain tasks in less time than conventional programs to a certain accuracy level. Fig. A-1, [Goodfellow'16], describes the relationship between artificial intelligence, machine learning, neural networks, and deep learning. Artificial intelligence is the science that encloses all the others as the superset.

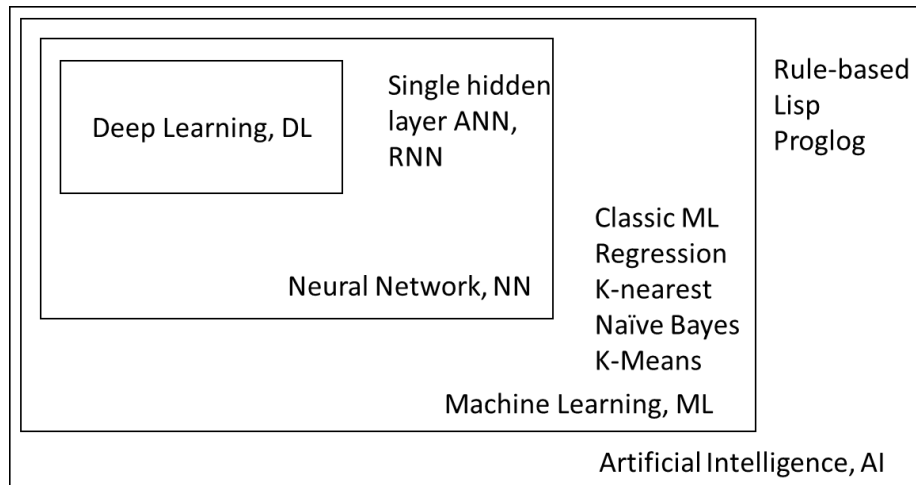


FIGURE A-1 Artificial Intelligence Categories Relationships [Goodfellow'16]

An artificial intelligence program that doesn't belong to machine learning is usually a hand designed program that models the knowledge and experience in the form of rules. Machine learning programs that don't belong to neural networks or deep learning depend on hand-

engineered features. Those features abstract the most important information from the knowledge and experience. Programs that are performing automatic features extraction are typically neural networks. Neural networks that have many hidden layers are said to be performing deep learning. Those deep learning programs usually work on huge data set. Fig. A-2, [Goodfellow'16], describes different artificial intelligence categories categorized based on features extraction.

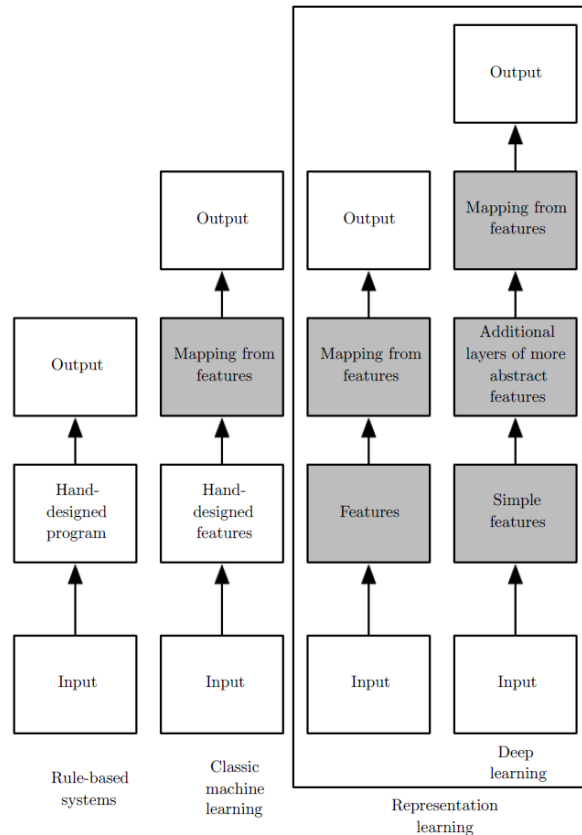


FIGURE A-2 Artificial Intelligence Categories [Goodfellow'16]

A.1 Rule-based Systems

Rule-based systems are hand-designed by engineers. In these systems, developers program past-experience, useful information, and facts in the form of rules. The rule-based system usually has an inference engine that infer new rules and information from given ones. Queries are used as input to the inference engine to draw conclusions based on the facts and rules provided to the engine. Though those systems had a lot of potential, they failed to solve complex problems. The most famous languages and inference engines are Prolog and Lisp.

Reinforced learning is another type of artificial intelligence where there are no predefined rules, however a rewarding-based learning system is established to learn the good behavior just like a child. Q-Learning, Fig. A-3, is an example of reinforced learning that beats pre-programmed rule-based system.

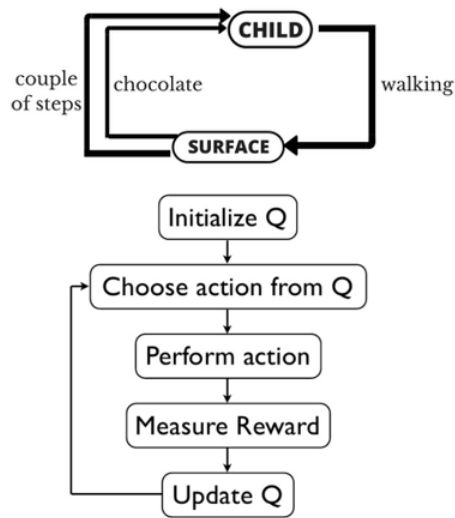


FIGURE A-3 Q-learning Reinforced Learning

A.2 Machine Learning, ML

A.2.1 ML Definition

Machine learning here means the classic machine learning techniques, artificial neural networks, and deep learning techniques. Goodfellow et al., [Goodfellow'16], formally define machine learning as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ".

The main keywords in the definition are tasks, performance, and experience. The program is said to be learning if its performance during performing a task improves by introducing new experience.

A.2.2 ML Tasks

The main tasks to be performed by machine learning techniques are:

i. Regression:

Regression task falls under supervised machine learning category. In this task, the program is trained on previous data to generalize the data trends and predict the system response given new system input. In the training phase, the previous data input, a.k.a. experience, together with the corresponding known system output are presented to the machine learning program. The program adapts its parameters to learn from previous experience and reduce the system error predicting new output.

ii. Classification:

Classification task also falls under supervised machine learning category. However, the program in this task is trained to classify given input into a known set of categories. Previous experience in addition to its known category is applied to the program during the learning phase. The program adjusts its parameters to reduce the error of classifying input data. After training phase, the program can classify new experience input into one of the known categories.

iii. Clustering

Clustering task falls under unsupervised machine learning category. In this task, the program is given experience as input only without specifying any output information. The program is trained to categorize input experience according to similar features they share into clusters. After training, the program is supposed to categorize new experience into an appropriate cluster according to its feature similarity with experience inputs belonging to this cluster.

iv. ML Experience

Experience, information, data, or machine learning input is usually represented in the form of vectors as Fig. A-4 shows. In case of supervised machine learning, each experience vector is associated with given output. Experience output can be represented in vector format as well. In classical machine learning techniques, important features are extracted from raw data before using them in either training or generalizing output given new experience.

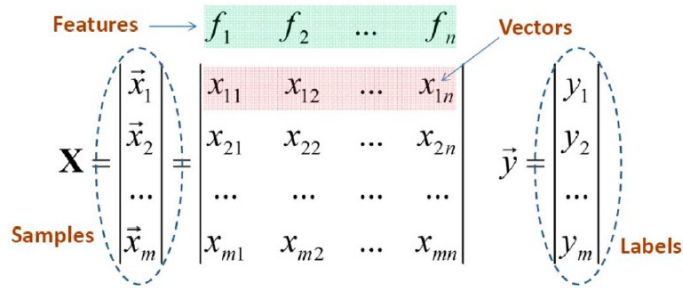


FIGURE A-4 ML Experience [Wang'17]

v. ML Performance

Just like any learning process, performance measure is crucial to monitor and adjust the learning process. The performance measure provides learning error measures to guide the learning process, it also provides testing error measures to evaluate the performance of the ML model. The most famous error rate is the mean square error MSE, Euclidian distance and variance. Mean square error is usually used with regression tasks.

$$\text{MSE}_{\text{test}} = \frac{1}{m} \sum_i (\hat{y}^{(\text{test})} - y^{(\text{test})})_i^2$$

Hit rate or 0-1 loss is another famous performance measure used with classification tasks. Performance measures can be carefully designed according to the performed ML task.

vi. Data Pre-processing

For all types of machine learning, data pre-processing is an important step to put raw input data in a suitable format for machine learning techniques. Standardizing on consistent data format is an essential step to ensure the learning process produces meaningful results. If the machine learning problem relates to image classification or clustering the all the images must be in the same image format, size, etc. If the machine learning is learning from raw data numbers, the numbers must have the same units standardized using the mean and standard deviation using the following formula.

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$$

Data scaling and normalization are also recommended to achieve better results for machine learning. This step avoids having one input variable dominating the other inputs.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Whenever the case of having some missing input data for any reason, there is a need to amend the missing using the mean, median or the most frequent of the rest of the data. Categorical data requires special data processing setup to convert them into number suitable for machine learning. Enumeration and one-hot encoding are usually utilized in this situation.

vii. Learning, Testing and Validation Data Set Distribution

Information data is usually divided into three portions. One for training purpose called the training dataset. Usually, the training dataset is 80% of the data size. The second dataset is the validation set, for validating the training process and adjusting its parameters. Usually, the validation dataset is 10% of the dataset size. The third dataset is the testing dataset. The testing dataset is used after the training phase to evaluate the machine learning program and its error rate. The testing dataset usually constitute the rest 10% of the original dataset.

viii. Capacity, Under-fitting and Over-fitting

One of the most important problems facing machine learning is matching the data complexity and features with the capacity of the learning ML program. A ML program that has more capacity than the data feature complexity results in a memorizing program that produces almost 100% error rate on the training data but produces very poor testing error rate. This is referred to the over-fitting problem.

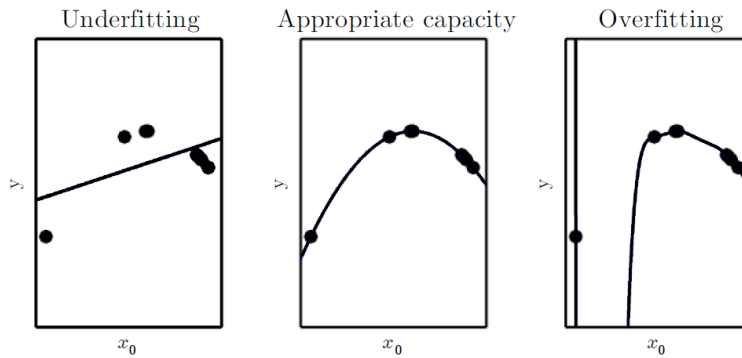


FIGURE A-5 Machine Learning Over-fitting and Under-fitting [Goodfellow'16]

The reason is that the ML program over fitted itself to the training data and lost the ability to generalize on the important features extracted from the training data. A ML learning program that has less capacity than the data feature complexity results in poor training rate and poor testing rate. This is referred to the under-fitting problem. The choice of appropriate capacity results in acceptable training error rate and testing error rate. It's said that the ML can generalize on the main features of the input dataset in this case.

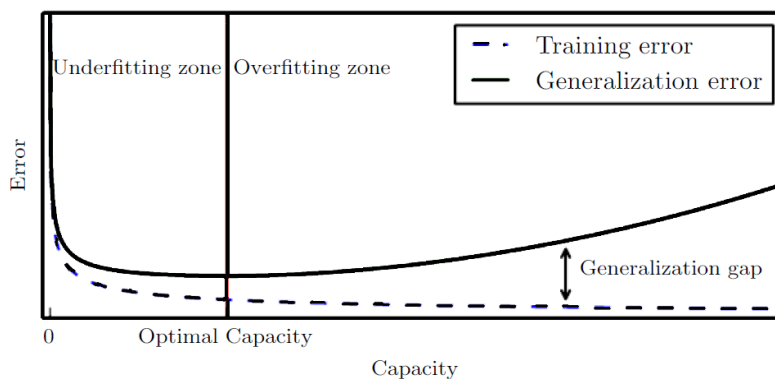


FIGURE A-6 Machine Learning Capacity versus Training and Generalization Error Rates [Goodfellow'16]

The above figure describes the relation between the capacity and the error rate. The increased capacity results in lower training error rate, and higher testing or generalization rate. This is typically called the generalization gap. The less capacity than appropriate results in larger training and generalization rates. The choice of the optimal capacity is crucial for a successful training and generalization process. However, the optimal capacity choice is a trial-and-error

process. It depends mainly on the experience of the ML program designer as well as some de-facto best practices.

A.2.3 Classic Machine Learning Algorithms

In classic machine learning, [Mitchell'97], the main features of data are hand-picked and designed by engineers. The main target of features extraction is to select the best features that abstract the dense information into simpler but important information. The quality of the hand-designed features maps directly to the quality of the classic machine learning programs that are produced. Engineers responsible for selecting the important features must have strong domain knowledge to be able to select high quality features.

The role of the classic machine learning program is to map and transform extracted features to the desired output. When new input is introduced to the classic machine learning program, the same features are extracted from the input data and those features are mapped to produce the output. The machine learning program can be designed to perform the same tasks of ML, regression, classification, or clustering tasks. There are several classic machine learning algorithms:

- Regression – Supervised Learning
 - Simple linear regression, multiple linear regression, and polynomial regression are famous classic machine learning technique. Fig. A-7, [Eremenko'17], shows the different types of regressions. The experience, input data x , and the known response y are used to find the parameters of the line/curve that will be used later to produce new response of new input data.

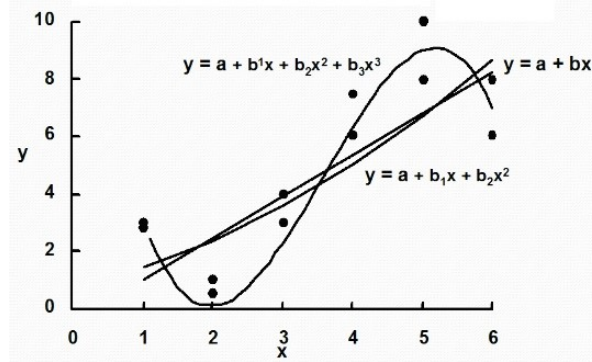


FIGURE A-7 Linear Regressions [Eremenko'17]

- Decision tree regression. Starts with plain dataset as in Fig. A-8. Information Entropy is utilized to determine data “splits” as in Fig. A-9. Decision tree is formed based on the splits as in Fig. A-10. Locate new instance leaf node. Calculate the average of nodes belonging to this leaf node

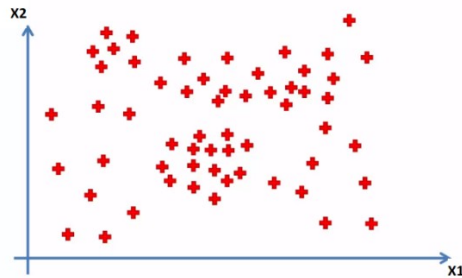


FIGURE A-8 Start with plain dataset [Eremenko'17]

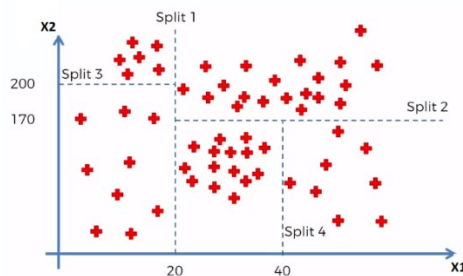


FIGURE A-9 Categorize data based on information entropy [Eremenko'17]

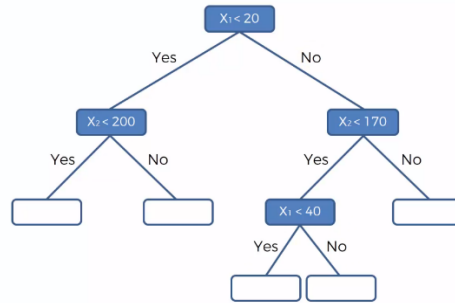


FIGURE A-10 Decision Tree formation [Eremenko'17]

- Other regression techniques are random forest regression and support vector regression.
- Classification – Supervised Learning
 - Support Vector Machine, SVM, is widely used in supervised classification problems. The goal of the SVM technique is to find the optimal hyperplane that separates dataset categories with maximum margins that guarantees an enhanced classification accuracy. Support vectors are created, Fig. A-11, between the initial hyperplane and the data points near it. A cost function is defined to maximize the margin between the hyperplane and the known categories of the dataset with help of the support vectors.

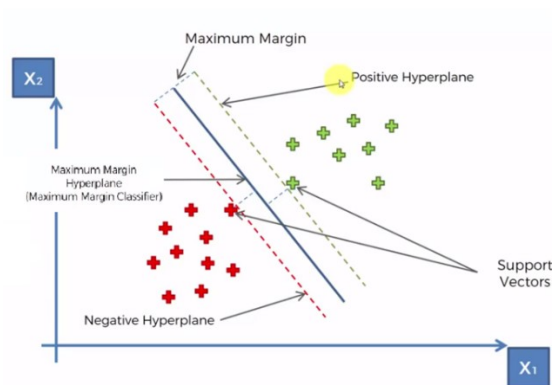


FIGURE A-11 Support Vector Machine in Classification [Eremenko'17]

- K-nearest Neighbor. Assuming there are two categories A and B and there is a need to classify a new data point as in Fig. A-12, the algorithm starts by selecting the number K of the neighbors then calculates the Euclidean distance of the new data point with all the existing points and takes the K nearest neighbors based on the calculated Euclidean distance. Among these k neighbors, the algorithm counts the number of the data points in each category. Finally, the algorithm assigns the new data point to the category that has a maximum number of the neighbors.

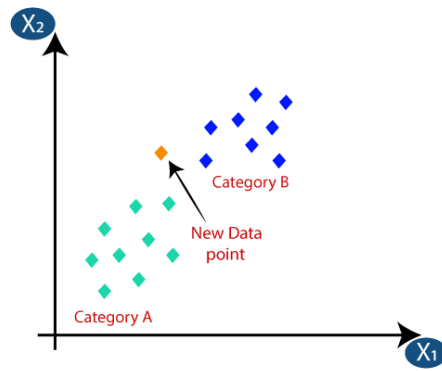
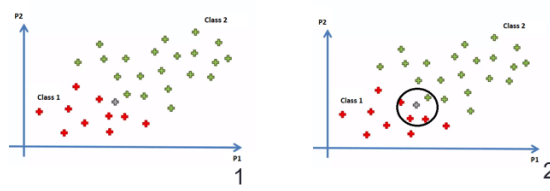


FIGURE A-12 K-nearest neighbor [Eremenko'17]

- Naïve Bayes. The algorithm starts with Class 1 and Class 2 already defined as in step 1 of Fig. A-13. To classify a new instance X , the algorithm draws a circle of radius r around the new instance, calculates $P(\text{Class1} | X)$ and $P(\text{Class2} | X)$, and finally assigns X to the class of higher probability.



$$P(\text{Class1}|x) = \frac{P(x|\text{Class1}).P(\text{Class1})}{P(x)}$$

$$P(\text{Class2}|x) = \frac{P(x|\text{Class2}).P(\text{Class2})}{P(x)}$$

FIGURE A-13 Naïve Bayes Classification Steps

- Other supervised learning techniques are logistic regression, decision tree classification and random forest classification.
- Clustering – Unsupervised Learning
 - K-Means Clustering. The algorithm starts with plain dataset as in step 1 of Fig. A-14, determine number of classes K, random pick of K centroids as in step 2, assign points to closest centroid as in step 3 based on constructing equi-distance planes to K centroids, then calculate new centroid of each cluster as in step 4, reassign points to new closest centroids as in step 5 then go to step 3, repeat until no reassignments of points occur to clusters as in step 6.

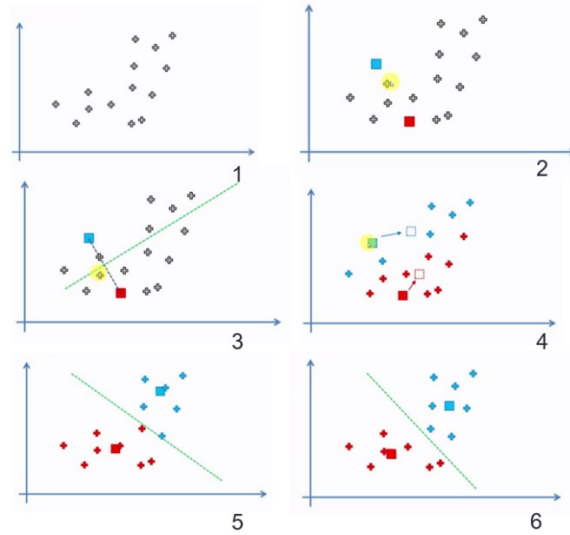


FIGURE A-14 K-Means Clustering Steps [Eremenko'17]

- Hierarchical Clustering. The algorithm starts by making each data point a cluster N points which results in N clusters as in step 1 of Fig. A-15. Then step 2 combines the closest two clusters in one cluster which results in $N-1$ clusters as in step 2. The algorithm keeps every clustering step in memory, and step 2 is repeated until everything point becomes in one cluster as in step 5. The number of clusters K is determined, and the algorithm goes back K steps in memory to get K -clusters formation as in step 6.

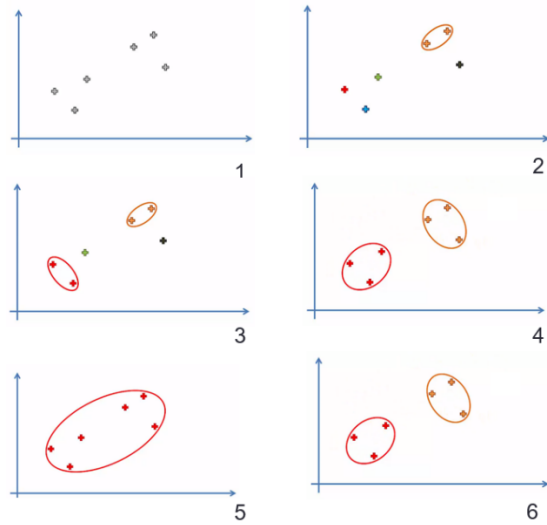


FIGURE A-15 Hierarchical Clustering Steps [Eremenko'17]

A.3 Artificial Neural Networks, ANN

Single perceptron is a simple summation function of all input signals multiplied by a weight value for each input. The summation is applied to what is called an activation function to produce the perceptron output. The ANN learning process is basically an optimization process to find the set of weights that minimize the training error rate. Once trained, the ANN structure and weights is ready to perform the regressions, classification or clustering task trained to do.

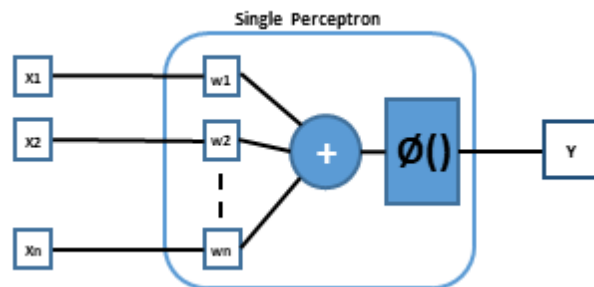


FIGURE A-16 Single Perceptron

$$Y = \phi\left(\sum_{k=0}^n x_k * w_k\right)$$

The choice of the activation function is an important design parameter for ANN success. The activation function can be as simple as a linear function, $\phi(x) = x$, or non-linear function. Non-linear like threshold function, rectifier function, hyperbolic function or sigmoid function allows

the ANN to learn non-linear problems and extract features that are hard to be extracted using classic machine learning techniques.

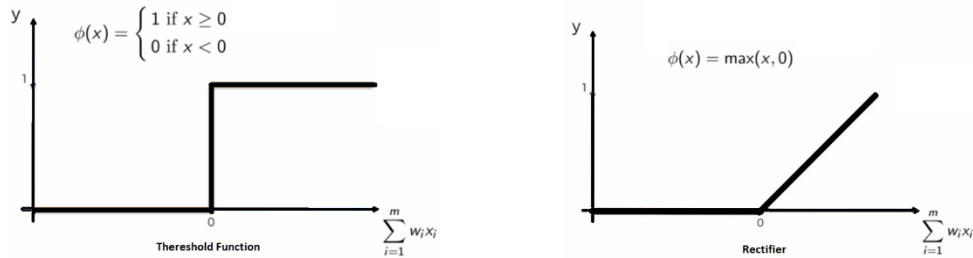


FIGURE A-17 Threshold and Rectifier Functions

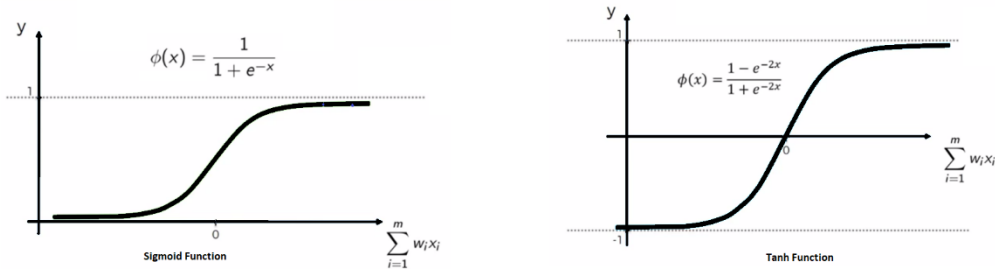


FIGURE A-18 Sigmoid and Hyperbolic Functions

Threshold activation function is usually used in output layer for ANN that performs classification tasks. Whereas the rectifier and sigmoid activation functions are usually used in the hidden layers. Hyperbolic activation function is usually used with ANN dealing with probability distributions. There are more non-linear functions used in research and in practice. Artificial neural network, ANN is composed of three layers: input layer, one hidden layer of many perceptron, and an output layer.

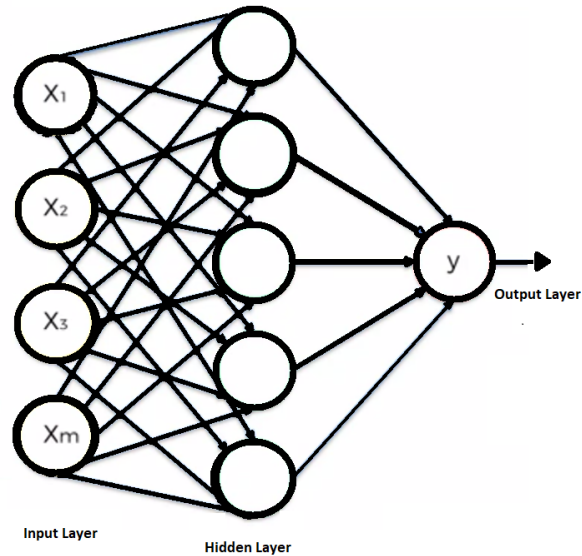


FIGURE A-19 Artificial Neural Network

A.3.1 Neural Networks Learning Algorithms

Back propagation is the most famous neural network learning algorithm. The algorithm initializes the different weights randomly then applies input to the ANN in the forward activation flow. The cost function is defined, and the error rate is calculated between the actual and desired values. The error is propagated back from the output layer to the input layers adjusting the different weights to reduce the gap between desired and actual output.

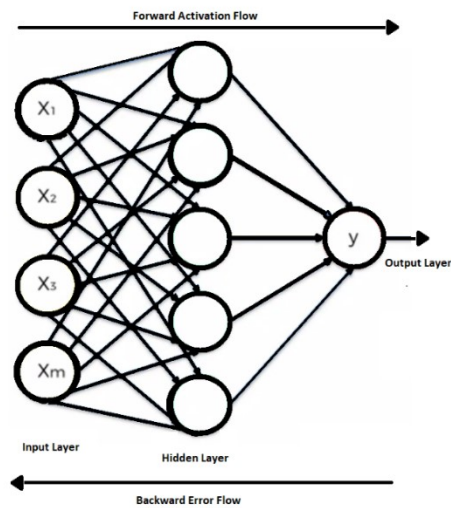


FIGURE A-20 Back Propagation Learning Algorithm

The process is repeated for all the input/output values in the learning dataset. The learning process converges as the gradient of the cost function descends and stabilizes to a minimum value as in Fig. A-21. The result of the learning phase is updated weight values that minimize the overall error rate across all the values of the learning set.

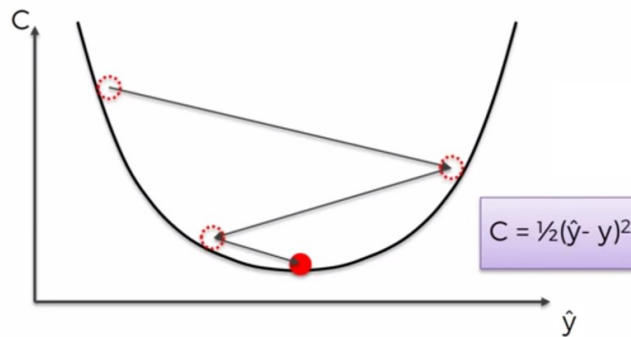


FIGURE A-21 Gradient descent algorithm [Eremenko'17]

A.3.2 Recurrent Neural Networks, RNN

The presented artificial neural network is a feedforward network. No output is fed back to neurons input. Thus, it's better suited for regression analysis and classification problems of independent datasets. When the input data is time series data, feedback from the neurons' outputs to neurons' inputs introduces a neural network that is capable of learning time series data.

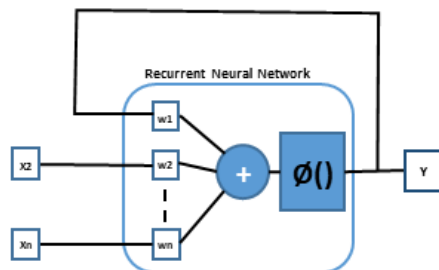


FIGURE A-22 Recurrent Neural Network

This type of neural networks is called recurrent neural networks. Recurrent neural network can learn data dependencies over time and generalize based on this information. Some researchers focused on optimizing the learning algorithms of RNN network as in [Williams'95].

A.3.3 Deep Learning, DL

The notion of deep learning refers to artificial neural networks that has several hidden layers of large number of neurons. The extremely large capacity of these networks can learn and extract larger number of small features. Deep learning depends on having huge dataset to be used in the training phase. This huge data is necessary to be able to find an optimized set of weights of these huge networks.

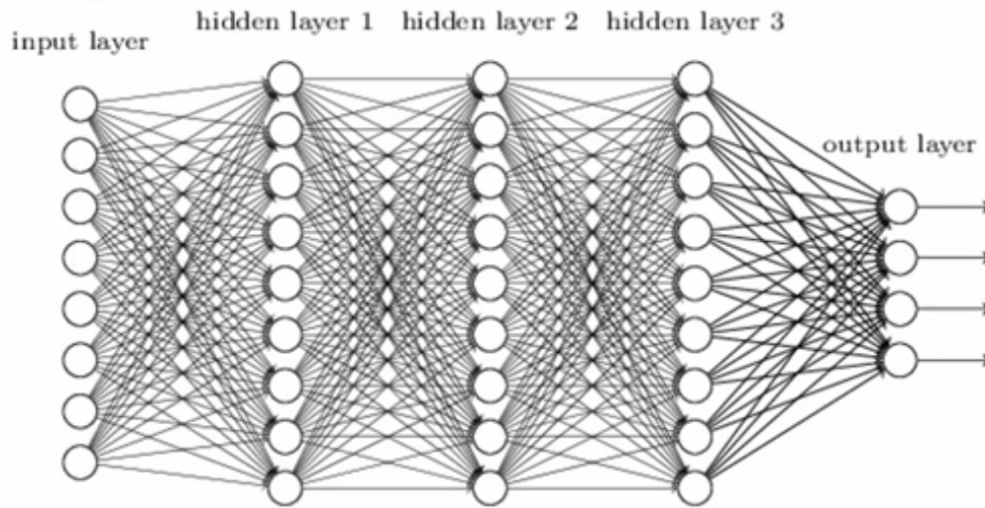


FIGURE A-23 Recurrent Neural Network

Deep learning became possible in recent years because of the availability of huge data, powerful processing machines, and advanced learning algorithms. There are many business driving forces that enforce a successful echo system that empowers deep learning.

A.3.4 Convolutional Neural Network, CNN

Convolutional neural networks are typically used in image recognition and classification problem. The input image is represented by 2D matrix as in Fig. A-24, then it's applied to a convolutional layer followed by a pooling layer. The role of convolutional and pooling layers is to reduce the size of the data representing the image by applying filters on the dataset without losing important features or losing the information about the dependencies between objects in the image. The convolutional layer is responsible for extracting the high-level features of the

image, there could be more than one convolutional layer in the CNN. On the other hand, the pooling layer is responsible for extracting the dominant features of the image. The output of the pooling layer is flattened before it gets applied to deep learning feed forward neural network. Referring to AI/ML categorization in Fig. A-2, each layer of the DL neural network extracts more features of the image at a certain level of abstraction.

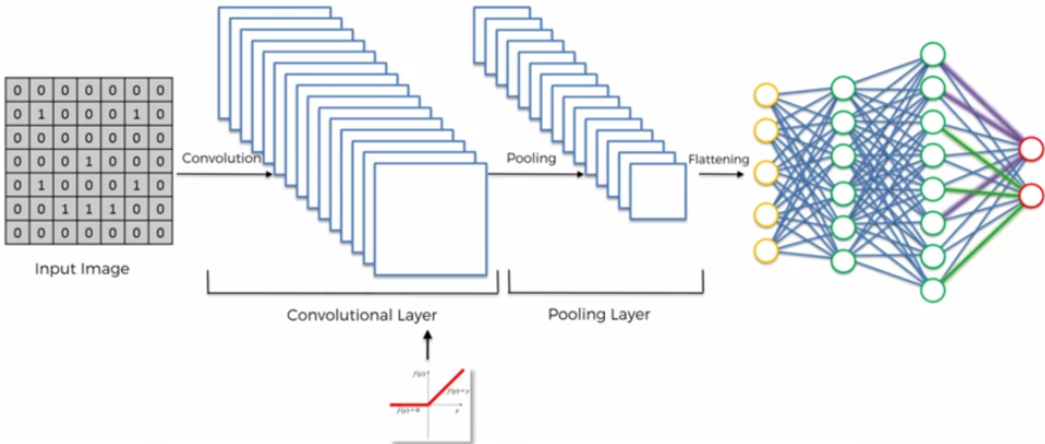


FIGURE A-24 Convolutional Neural Networks, CNN [Eremenko’17]

A.3.5 Autoencoders

Feed-forward neural network is used to encode large data into a smaller set of parameters representing the original data. The core idea of Autoencoder is to train a deep neural network to produce an output that matches the input data to a certain degree of error, [Goodfellow’16] and [Zhang’18]. The output layer must have the same size of the input layer. If the Autoencoder middle layer size is smaller than the number of inputs as shown in Fig. A-25, then the Autoencoder can perform dimensionality reduction of data.

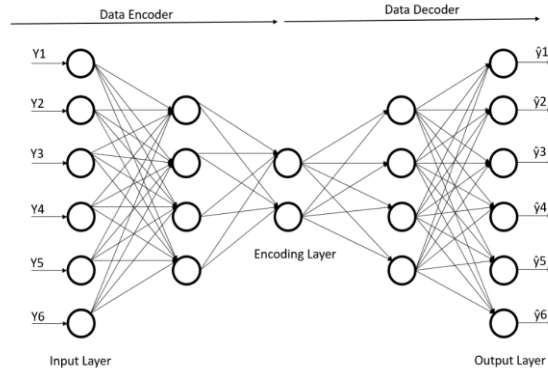


FIGURE A-25 TRAINING of DEEP LEARNING AUTOENCODER

After training, the autoencoder is split into two models at the middle layer: an encoder model and a decoder model. Fig. A-26 shows the trained Autoencoder after being split into the two models. The layers from the input layer to the middle layer represent the encoder model. When the original data is presented as input to the encoder model, the output values of the middle layer neurons are the encoded, compressed data of the original data. Therefore, we refer to number of neurons in the middle layer as the number of encoding parameters. The layers from the middle layer to the output layer represent the decoder model with the middle layer playing as the input layer to the decoder model.

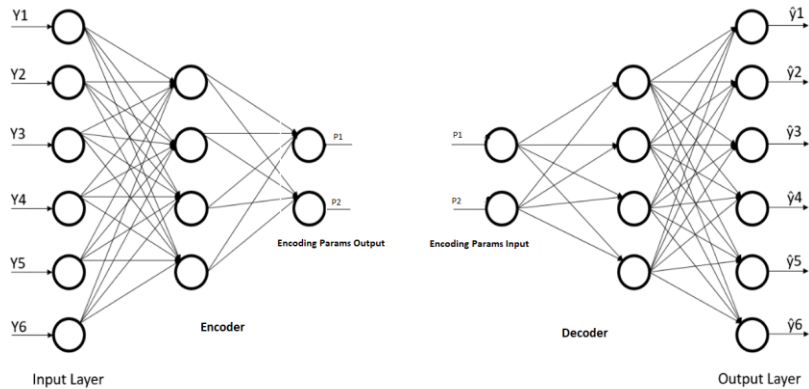


FIGURE A-26 AUTOENCODER Split to Encoder and Decoder

The compressed encoded data – the output of the middle layer, are presented as input to the first decoder model layer to be decoded to reproduce the original data within certain degree of error.

Encoding large data points into a smaller number of parameters can be used in performing the

required waveform data compression. The decoder model must be provided with the compressed data to be able to decompress and restore the original data.

A.4 Machine and Deep Learning Applications

ML and DL learning algorithms can generally address problems of type:

- Classification
- Regression (Modeling, Prediction)
- Clustering
- Anomaly Detection
- Dimensionality Reduction

ML and DL techniques are typically used when there is no conventional algorithm to solve a problem with adequate accuracy is adequate time, or accuracy and performance of conventional algorithms deteriorates don't scale up with increased data size and problem complexity. Classic ML techniques are typically used with when there are small or medium training data, and the domain knowledge is known such that it's extracting reliable and important features can be achieved. DL techniques are typically used when there is large training data and less domain knowledge to extract the important features.

APPENDIX B

Model Order Reduction of Non-Linear Systems using Recurrent Neural Networks

B.1 Overview

In the proper orthogonal model technique, the most important modes are selected to perform model order reduction based on the singular value decomposition analysis. The main hurdles of the model order reduction using this method is the non-linear term. This non-linear term makes the computation expensive even if we select few modes for the model order reduction. The more modes we must include the more computation power and time it needs to calculate the time dependent factors that complete the reduced order model definition. The proper orthogonal model-based model order reduction depends on having high fidelity measurements of the actual system or simulation results from detailed full order model that takes huge efforts to develop and simulate. Given that data, our approach is to train a recurrent neural network to learn the system behavior and hence represent a reduced order model of the system.

Using RNN to model non-linear systems doesn't require explicitly selecting certain modes and sacrificing other upfront. The RNN learning mechanism is responsible for learning important modes and ignoring others to minimize its cost function. The RNN topology, the learning algorithm and the dataset will define the accuracy of the reduced model.

To obtain training data, we used Matlab to implement exact solution of selected non-linear systems. The data obtained from the exact solution is used twice. Once to create a POD based reduced order model using the first couple of important modes, and the second time to train our RNN to produce neural network based non-linear reduced order model. The RNN topology we selected looks like the MIMO RNN we proposed in section 5.2. We treated the initial values at

different grid points of the x dimension as the multi-inputs for the non-linear system. In addition, we treated the system output at different points of the grid as the multi-outputs. As the time advances, the RNN learns the expected multiple output at each point of the x axis grid.

The main contributions of this research are:

- Applying RNN modeling techniques to non-linear systems.

B.2 One-dimensional non-linear Schrodinger equation

The first selected non-linear system is the one-dimensional non-linear Schrodinger like wave equation with x and t independent variables:

$$i \frac{\partial}{\partial t} (u(x, t)) + \frac{1}{2} \frac{\partial^2}{\partial x^2} (u(x, t)) + |u(x, t)|^2 u(x, t) = 0$$

Re-writing the equation:

$$\frac{\partial}{\partial t} (u(x, t)) = \frac{i}{2} \frac{\partial^2}{\partial x^2} (u(x, t)) + i |u(x, t)|^2 u(x, t)$$

Utilizing Fourier basis, we get:

$$(\hat{u}_t) = \frac{-i}{2} k^2 (\hat{u}) + i * \text{fft} (|u|^2 u(x, t)) ; k \text{ is } (2*\pi/L)*[0:n/2-1 -n/2:-1] :$$

Where L is the length of x dimension and n is the number of discrete points taken along the length L. We can utilize Matlab ode45 function to solve the above equation. The initial value can take any shape, we selected, $\text{sech}(x)$, $2\text{sech}(x)$, and $\text{sech}(x^2)$ to study the neural network behavior at different conditions. This Matlab solution generates the non-linear system data that is used to perform POD-based model order reduction and our proposed S-RNN model order reduction.

```
init_vals = sech(x); ut = fft(init_vals);
t=linspace(0,2*pi, 41); k = (2*pi/L)*[0:n/2-1 -n/2:-1].';
[t, utsol] = ode45('nls_rhs',t,ut,[],k);
```

The Matlab function nls_rhs is defined as:

```
function rhs= nls_rhs(t,ut,dummy,k)
    u = ifft(ut);
    rhs=(-i/2)*(k.^2).*ut + i*fft((abs(u).^2).*u);
end
```

The solution is retrieved by doing inverse fft to utsol;

```

for j=1:length(t)
    usol(j,:) = ifft(utsol(j,:));
end

```

The solution can be plotted using:

```

surf(x, t, abs(usol)), shading interp;

```

B.3 POD-based reduced order model

The solution data of the full system is contained in the `usol` array. As the size of the system increases the computation power increases. It's required to get a reduced size model of the full system. Using the data obtained from full model system simulation or actual system measurements, we can utilize the proper orthogonal decomposition method to get a reduced order model. Section 2.4 outlined the steps required to perform POD-based reduced order model.

1. The first step is to perform the singular value decomposition matrices. Using Matlab `svd()` function this can be calculated as:
2. `[u,s,v] = svd(usol, 'econ');`
3. The singular matrix `s` is a diagonal matrix ordered by values. The matrix is examined, and the most important number of modes are identified.
4. The corresponding number of vectors are selected from the basis matrix `u`. Selecting the first couple of modes can be done using Matlab as: `phi=u(:,1:2);`
5. Taking advantage of the orthonormal basis of the matrix `u`, and applying the general equation mentioned in section 2.4 to the wave equation problem:

$$\frac{d}{dt}(a(t)) = \phi_r L \phi_r * a(t) + \phi_r N(\phi_r * a(t))$$

B.4 RNN Model Order Reduction Results

B.4.1 Experiment 1: Sech(x) as an initial function:

B.4.1.1 Full System Solution Using Matlab

Using `sech(x)` as initialization function as an input to the Matlab `ode45()` solution outlined above, produces the below solution. It's clear that there is a single mode dominating the waveform.

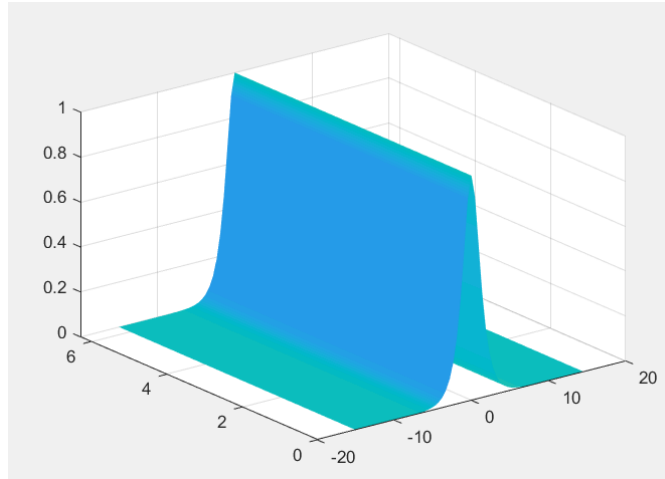


FIGURE B-1 Matlab Simulation with $\text{Sech}(x)$ initial function

The single mode observation is confirmed by performing singular value decomposition. The following figure plots the diagonal values divided by their summation. There is only one value dominating the diagonal matrix as expected.

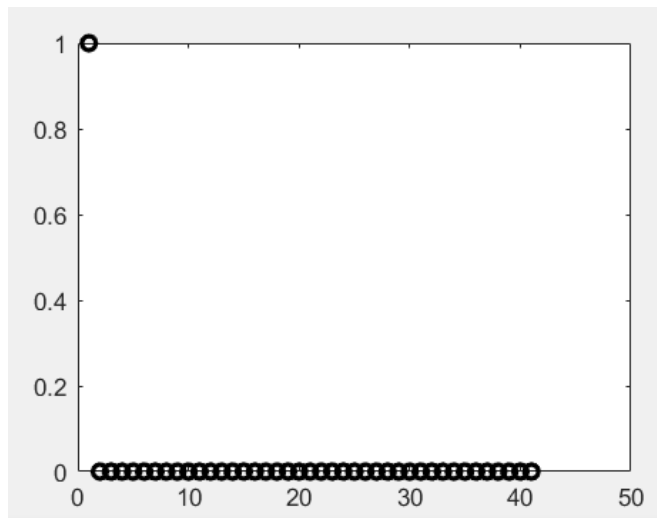


FIGURE B-2 SVD Analysis

The first three modes of the matrix u are plotted in the following figure. The first mode in blue is the dominating mode, while the rest have negligible contribution and are considered numerical errors.

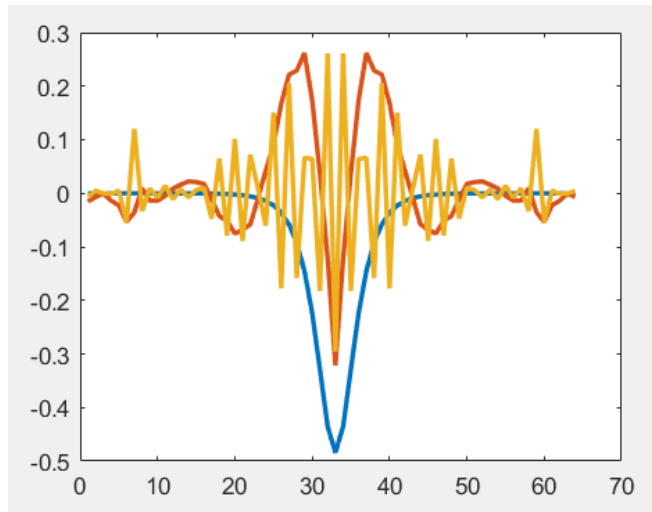


FIGURE B-3 First three modes plot

B.4.1.2 POD-based MOR with 1 Mode

Selecting only one mode to perform model order reduction produced satisfactory results.

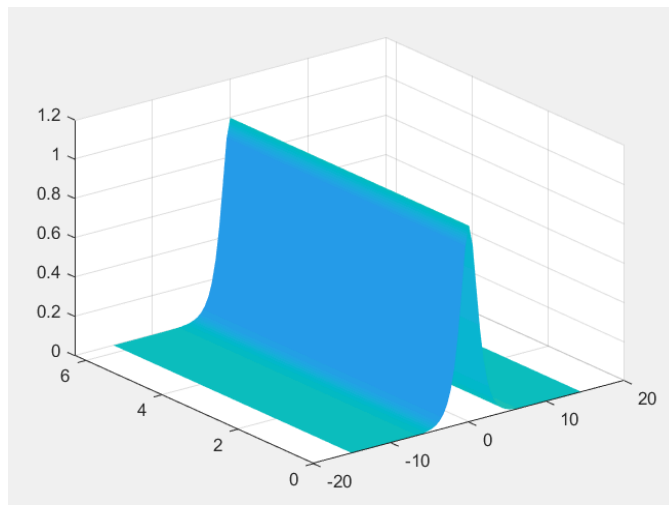


FIGURE B-4 One Mode POD MOR

B.4.1.3 S-RNN MIMO of 2nd Order MOR

Our implementation of model order reduction using MIMO is utilized to perform machine learning based model order reduction. The grid values of the solution at the x dimension represent the multi-input values. The output values are the system response after one time iteration. During the learning phase the RNN is fed with the usol data obtained from the exact

Matlab solution. As time advances, the RNN learns the system behavior. After only 9 iterations, the RNN learned the system behavior with a very good MSE of $1.48e-11$.

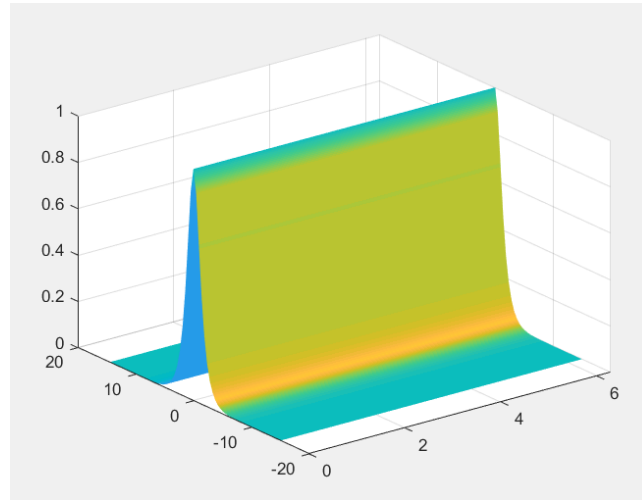


FIGURE B-5 Second order S-RNN MOR

The trained RNN is then subjected to the same initial value function, to produce similar results to Matlab exact solution and the POD-based reduced model order solution.

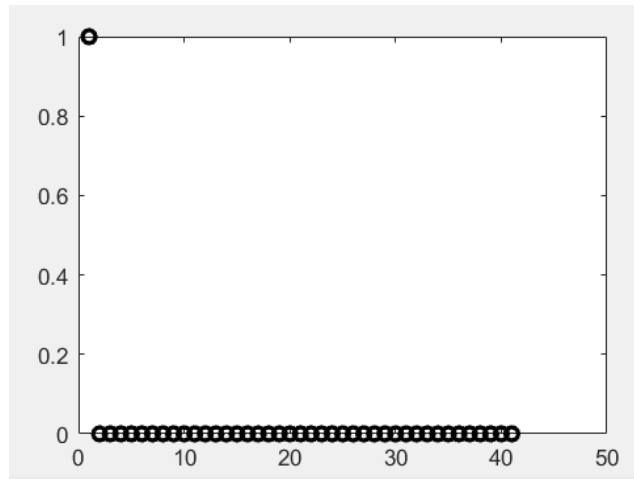


FIGURE B-6 SVD analysis of Second order S-RNN solution

To double check the dominating modes of RNN reduced order model response, we perform singular value decomposition analysis to RNN solution data array. Plotting the singular diagonal matrix values divided by their summation confirmed that there is only one mode dominating the RNN system response as expected.

B.4.2 Experiment 2: $2*\text{Sech}(x)$ as an initial function:

B.4.2.1 Full System Solution Using Matlab

Applying a different initialization function to the same wave equation produced the following waveform solution using Matlab `ode45()` function. It's expected that more than one mode contributes to the full system solution.

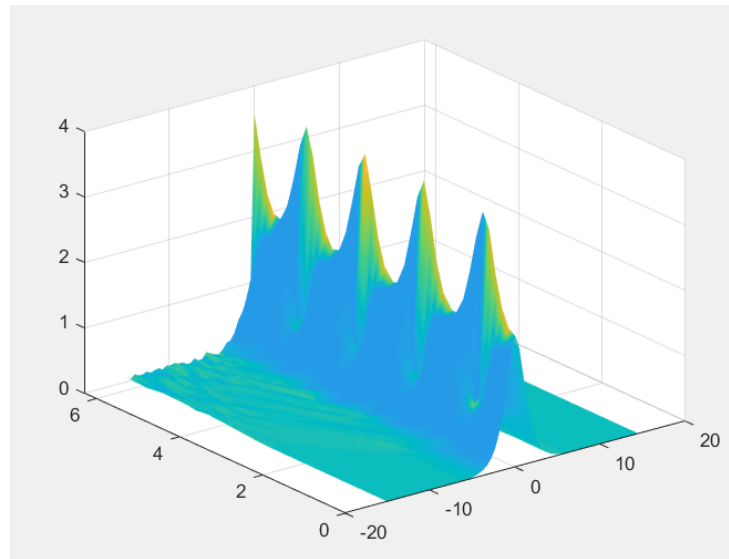


FIGURE B-7 Matlab Simulation with $2*\text{Sech}(x)$ initial function

Performing singular value decomposition analysis on the `usol` matrix confirms this observation as clear in the following figure. The relative value of the singular values diagonal matrix shows that there are two main modes dominating the full system behavior.

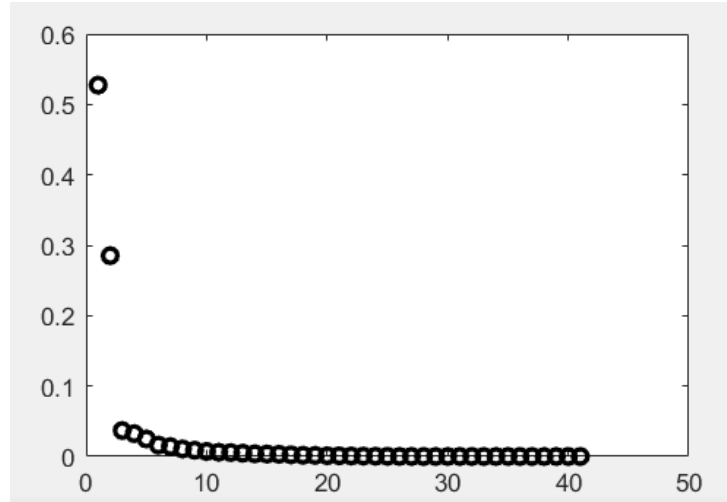


FIGURE B-8 SVD analysis of Matlab solution

The following figure shows the first three modes of the matrix u . The first mode is the blue curve, the second is the red curve. The third mode is less significant mode compared to the first two and considered numerical errors.

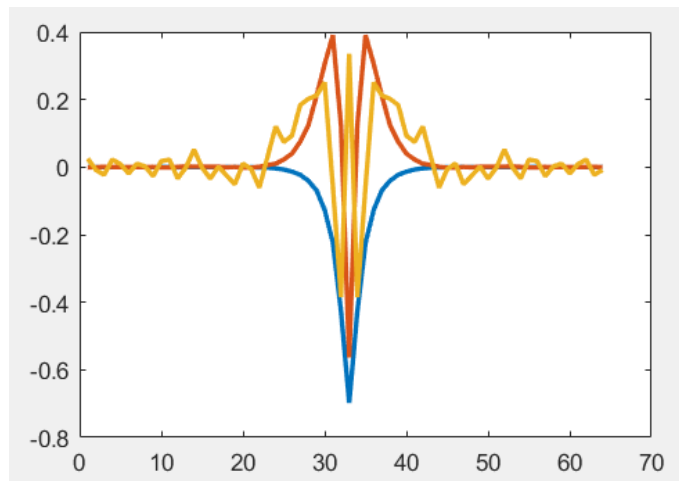


FIGURE B-9 Mode plot Matlab solution

B.4.2.2 POD-based MOR with 2 Modes

Selecting the first two modes and performing POD-based reduced model order reduction resulted in a solution that has the following waveform.

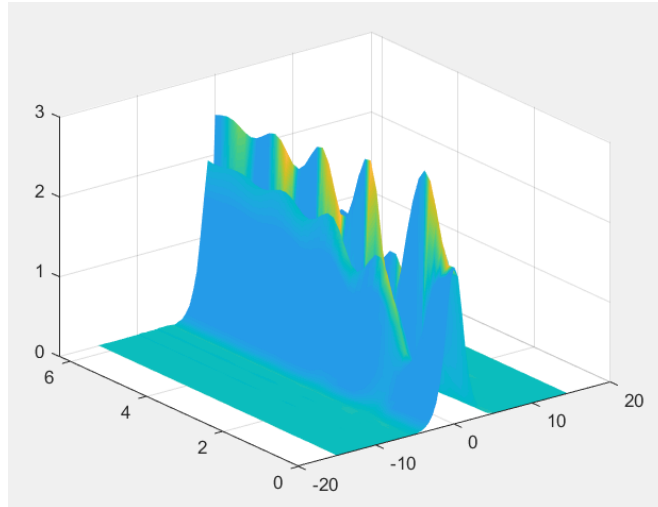


FIGURE B-10 Matlab simulation of POD solution

B.4.2.3 S-RNN MIMO of 2nd Order MOR

Our proposed MIMO RNN of order 2 is used to model this non-linear full system response. The training phase stopped after 122 iterations with MSE of 0.0133. Plotting the RNN response to the same initialization function showed that the results are not satisfactory. The RNN reduced order model didn't manage to model important system behavior. Performing singular value decomposition and plotting the relative values of the singular matrix diagonal values shows that the RNN response model almost only one mode.

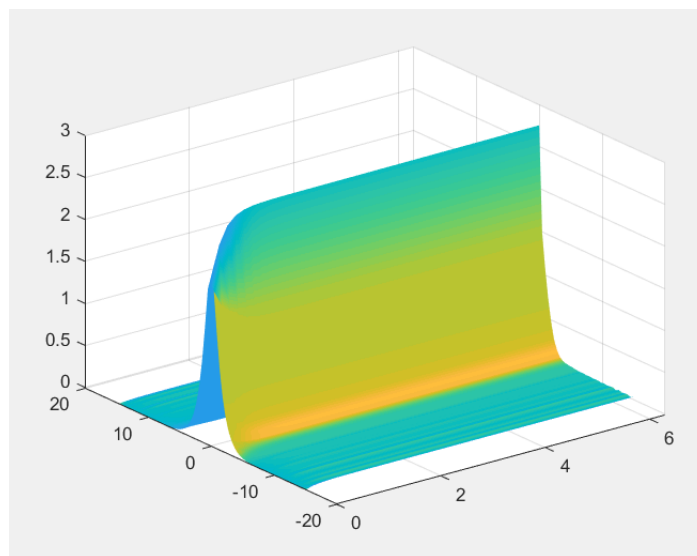


FIGURE B-11 Second-order S-RNN MOR

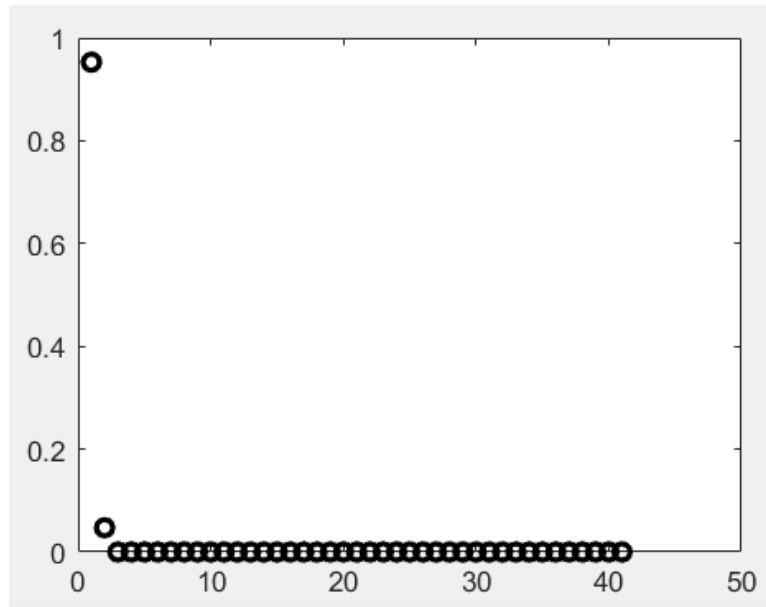


FIGURE B-12 SVD analysis of S-RNN MOR

B.4.2.4 S-RNN MIMO of 5th Order MOR

Raising the order of the RNN MIMO order to 5 produces satisfactory results with better MSE of 0.00183 after 1000 iterations. Plotting the RNN response of the trained network shows that the S-RNN reduced order model was able to retain important system characteristics.

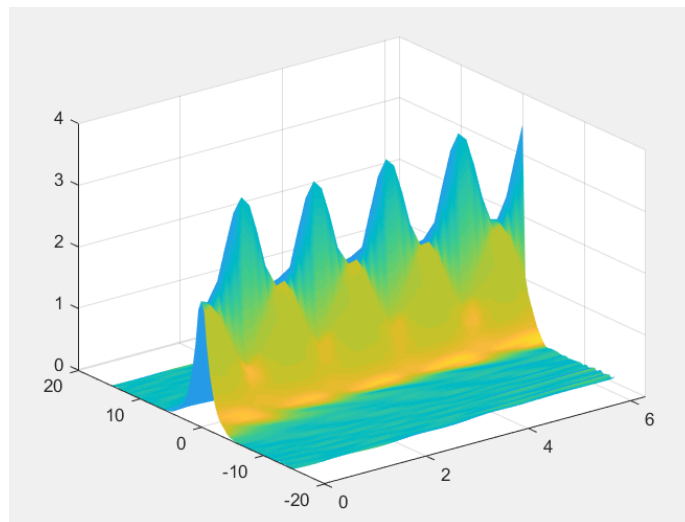


FIGURE B-13 5th-order S-RNN MOR

Performing SVD and plotting the relative singular matrix diagonal values shows that the RNN response has 2 dominating modes like the full system response.

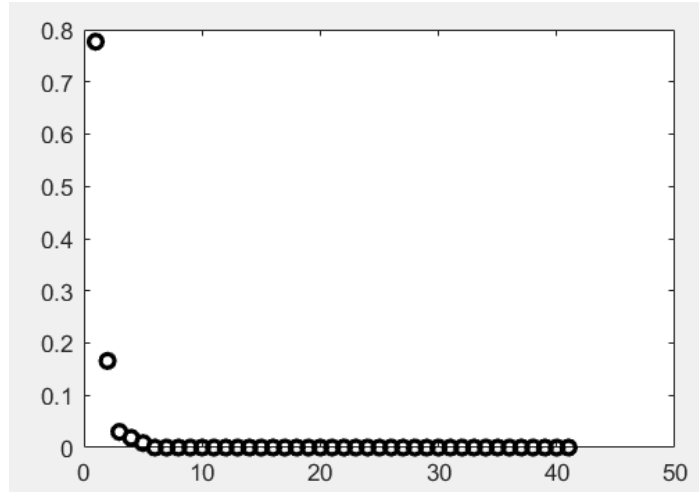


FIGURE B-14 SVD analysis of 5th-order S-RNN MOR

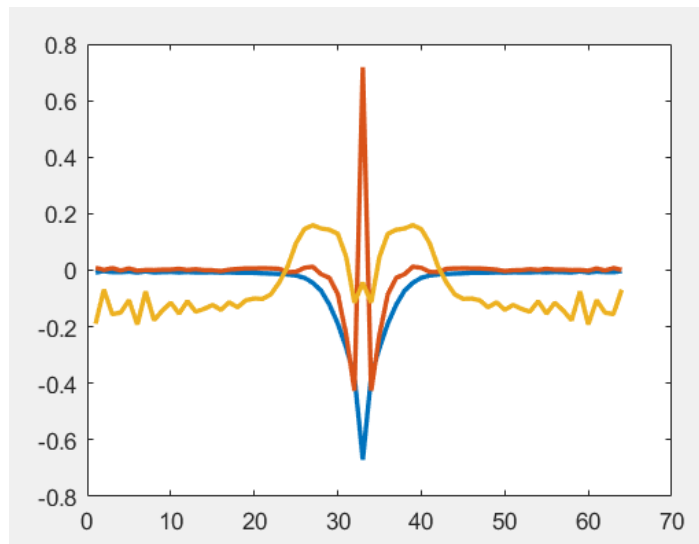


FIGURE B-15 5th-order S-RNN MOR mode plot

B.4.3 Experiment 3: $\text{Sech}(x^2)$ as an initial function:

B.4.3.1 Full System Solution Using Matlab

Applying $\text{sech}(x^2)$ as an initialization function to the same system produces different response. The full system response in the following figure shows more involving characteristics. It's expected that more than 2 modes will dominate the system behavior.

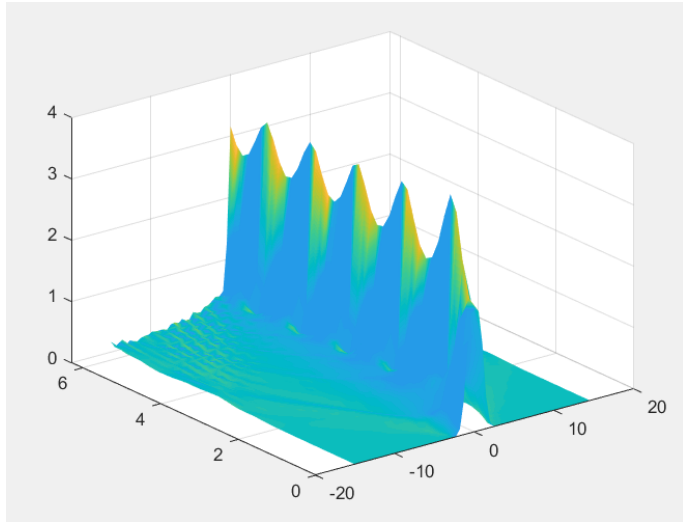


FIGURE B-16 Matlab Simulation with $\text{Sech}(x^2)$ initial function

The singular value decomposition analysis confirms the observation, more than two modes are contributing to the system behavior.

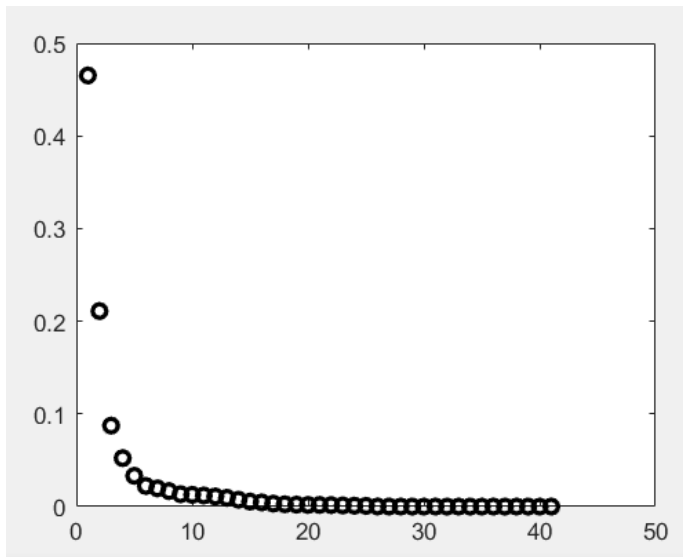


FIGURE B-17 SVD analysis of Matlab solution

The following figure plots the first three dominating modes.

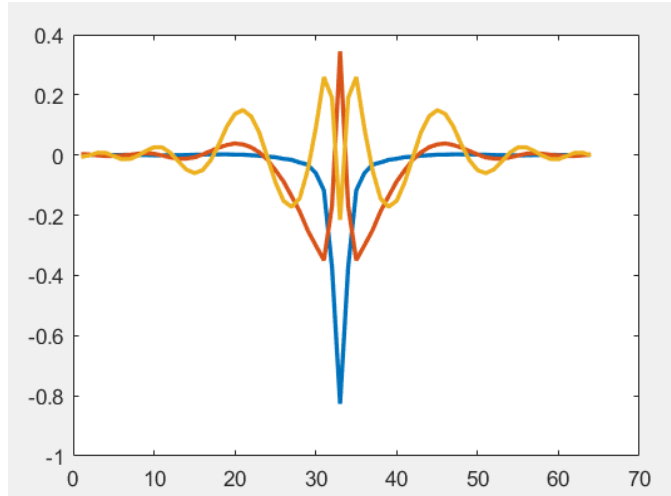


FIGURE B-18 Mode plot of Matlab solution

B.4.3.2 POD-based MOR with 2 Modes

Utilizing two modes to perform POD-based singular value decomposition reduced order model results in losing important system behavior. The following figure shows the POD-based reduced order model using only two mode.

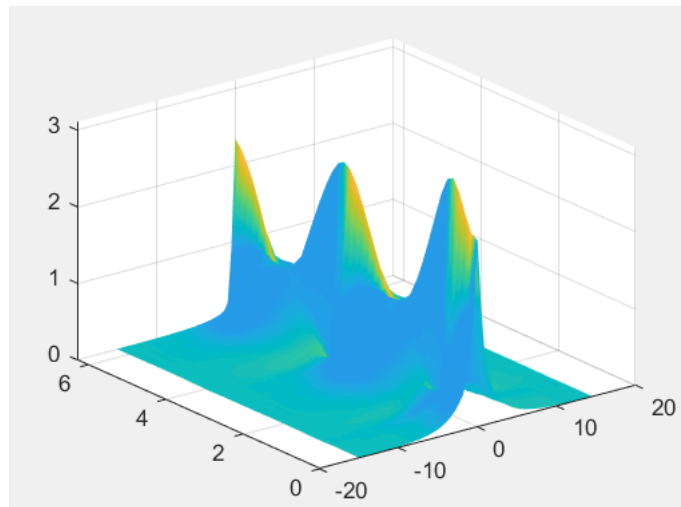


FIGURE B-19 POD analysis with 2 modes

Using more than two modes to perform POD-based model order reduction requires more expensive processing power.

B.4.3.3 S-RNN MIMO of 3rd Order MOR

Using our proposed MIMO based RNN of order 3 to be trained on the full system response consumed 1000 iterations and stopped at MSE of 0.00590. However, plotting the RNN response to the same initialization function showed that the reduced order model lacked important system characteristics.

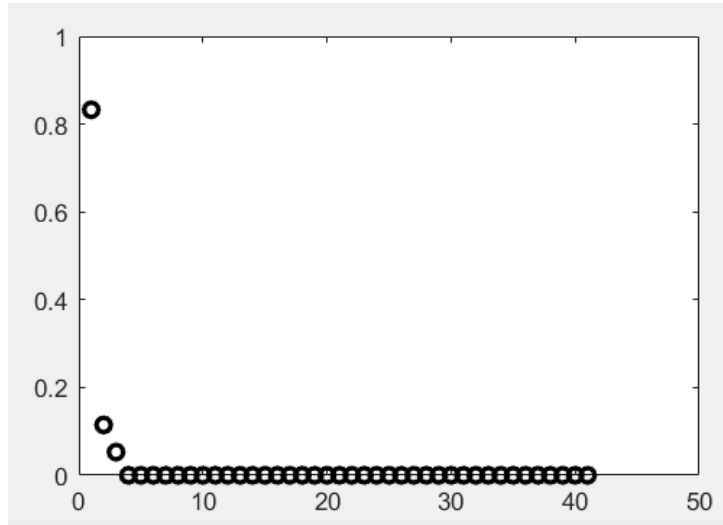


FIGURE B-20 SVD Analysis of POD solution

The singular value decomposition analysis of the RNN system response confirmed these observations. The trained RNN was able to capture only three modes as shown in the following figure.

B.4.3.4 S-RNN MIMO of 5th Order MOR

Increasing the order of the S-RNN MIMO model order to 5 resulted in better results. The training phase finished after 1000 iterations with better MSE of 0.000876. The trained RNN response to the same initialization function resulted in similar behavior to the full system.

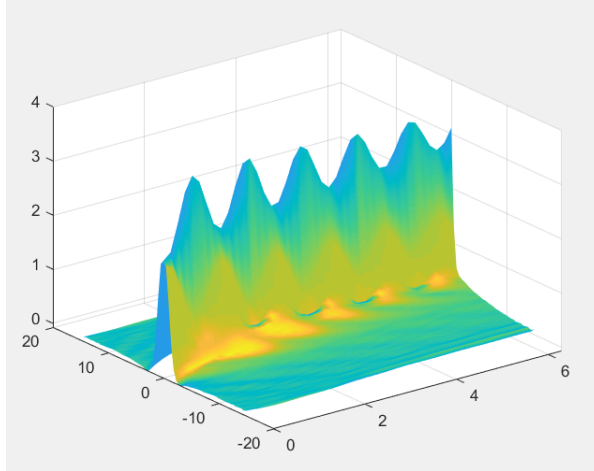


FIGURE B-21 5th order S-RNN solution

The singular value decomposition analysis of the RNN response shows 4 dominating modes.

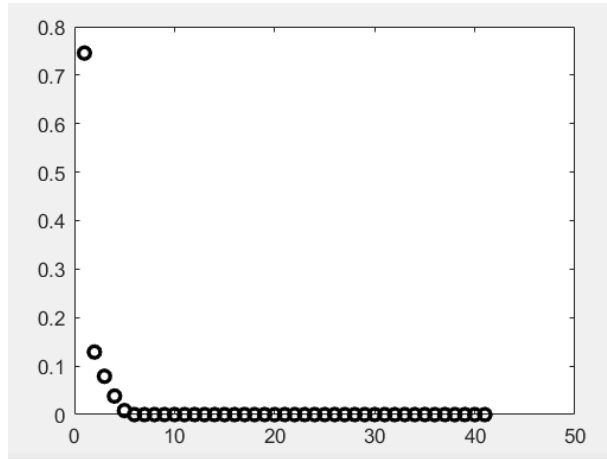


FIGURE B-22 SVD analysis of 5th order S-RNN solution

APPENDIX C

Detailed trained models result

C.1 Autoencoding fixed time 1000-points ECSM Waveforms Detailed Results

Similar measures are used to evaluate the performance of the Autoencoders trained to encode 1000-points sampled ECSM waveforms. Two encoding parameters give mean correlation coefficient greater than 0.999 and less than 1% average percentage error with a compression ratio 104.06. Fig. C-1 shows reconstructed rising and falling-edge waveforms after encoding with 2 parameters and model ID#2. Tables C.1 and C.3 list the best encoding results at different encoding parameters. Tables C.2 and C.4 show the SVD compression results at different Sigma rank number. The 2-parameters Autoencoder model gives 3.37x better compression ratio than the nearest SVD compression results at sigma rank equals 8. However, SVD runtime decoding is 14x faster than Autoencoders decode time.

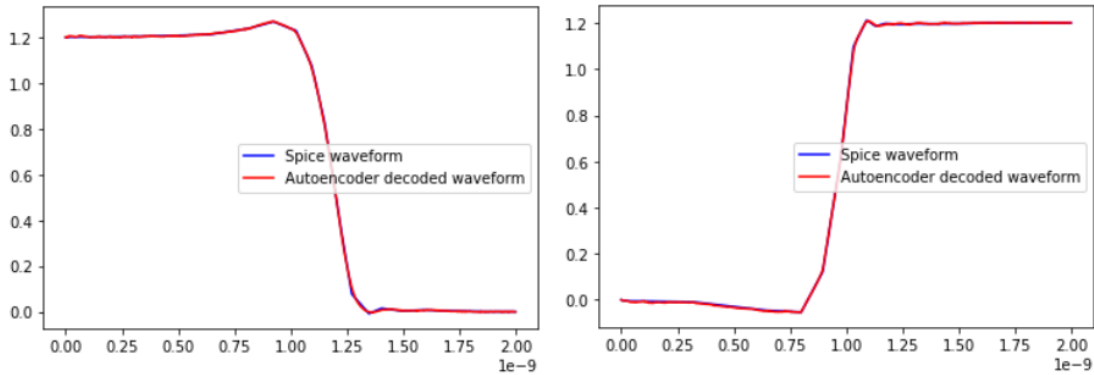


FIGURE C-1 DECODED ECSM FALLING AND RISING-EDGE WAVEFORMS VS. SPICE WAVEFORMS

Table C.1 Autoencoding 1000-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	3	2	2	2
Decoder size (KB)	4320	4249	4249	4249
Avg % error 0.8VDD	0.2936	0.3169	0.4676	0.5362
Avg % error 0.5VDD	0.1957	0.1962	0.3755	0.3318
Avg % error 0.2VDD	0.2383	0.2425	0.4363	0.4247
Mean corrccoef	0.99998	0.99997	0.99992	0.99987
Model Loss (MSE)	1.46E-06	1.60E-06	4.10E-06	5.9E-06
Compression Ratio	42.13	64.06	86.13	104.06

Table C.2 SVD 1000-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number

Falling-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.1599	1.9487	0.6635	11.7535	3.1479	99.8715	10.8588	386.27
Avg % error 0.5VDD	0.1314	1.6828	0.4421	7.5705	1.2438	47.2797	3.90126	69.87
Avg % error 0.2VDD	0.1675	3.3921	0.9593	17.5298	4.3191	74.2143	13.1785	190.324
Mean corrcoeff	0.99993		0.99908		0.99313		0.97136	
Model Loss (MSE)	6.2216E-06		8.294E-05		6.3932E-4		2.626E-3	
Compression Ratio	31.25		62.5		125		250	

Table C.3 Autoencoding 1000-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Rising-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	2	4	2	2
Decoder size (KB)	4320	4249	4249	4249
Avg % error 0.8VDD	0.2476	0.2925	0.3005	0.3356
Avg % error 0.5VDD	0.1915	0.2105	0.2063	0.2451
Avg % error 0.2VDD	0.2927	0.3202	0.3153	0.3732
Mean corrcoeff	0.99997	0.99996	0.99996	0.99995
Model Loss (MSE)	1.75E-06	2.50E-06	2.60E-06	3.8E-06
Compression Ratio	42.35	62.52	86.13	104.06

Table C.4 SVD 1000-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number

Rising-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.4669	8.9164	2.2491	26.4061	6.9245	92.2892	17.3397	234.75
Avg % error 0.5VDD	0.1877	4.3850	0.6449	10.6402	1.4726	57.5539	4.6693	168.95
Avg % error 0.2VDD	0.3235	7.1173	1.6261	26.5724	5.7295	36.5229	14.7416	88.2501
Mean corrcoeff	0.99959		0.99634		0.98484		0.95517	
Model Loss (MSE)	3.663E-5		3.2E-4		1.351E-3		4.006E-3	
Compression Ratio	31.25		62.5		125		250	

C.2 Autoencoding variable time 50,100,150-points ECSM Waveforms Detailed Results

C.2.1 Falling-edge 50-samples

Table C.5 Autoencoding 50-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	16	15	15	15
Decoder size (KB)	75	39	39	39
Avg % error 0.8VDD	1.9486	2.0433	1.8758	2.316
Avg % error 0.5VDD	1.9841	1.9792	1.861	2.1428
Avg % error 0.2VDD	1.9922	2.0084	1.8777	2.0736
Mean corrcoeff	0.99995	0.99990	0.99982	0.9996

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
		4		
Model Loss (MSE)	2.99E-06	6.09E-06	1.24E-05	3.87E-05
Compression Ratio	3.11	6.19	12.28	24.15

Table C.6 SVD 50-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number

Falling-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.3278	42.53	0.3117	118.001	0.3942	5.4991	0.7765	15.1329
Avg % error 0.5VDD	0.1809	9.0276	0.1878	5.9907	0.19088	1.9743	0.4008	2.4779
Avg % error 0.2VDD	0.1073	3.9161	0.1171	1.5579	0.1590	2.1628	0.4604	26.6131
Mean corcoef	0.99999		0.99993		0.99951		0.99586	
Model Loss (MSE)	9.513E-8		1.889E-6		1.5075E-5		1.53E-3	
Compression Ratio	1.56		3.125		6.25		12.5	

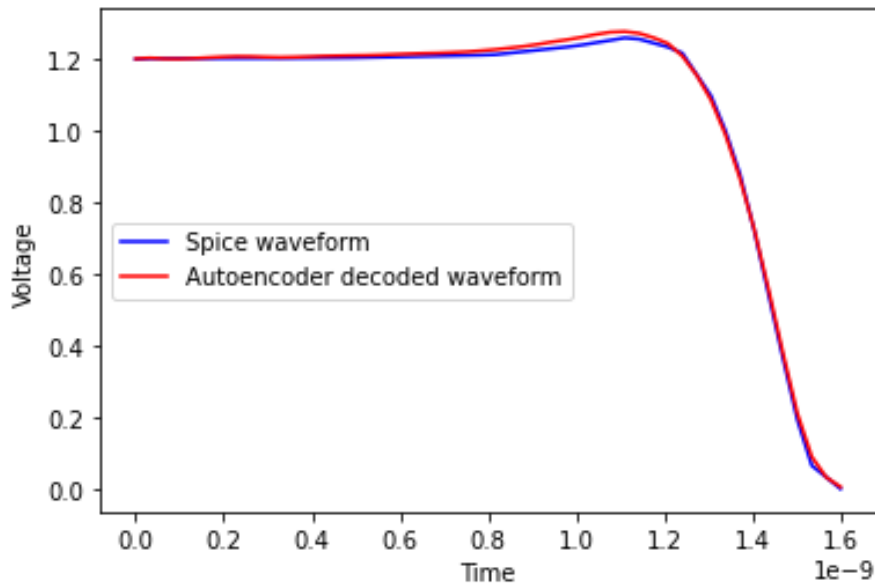


FIGURE C-2 SPICE vs reconstructed 50-points sampled falling edge waveform

C.2.2 Rising-edge 50-samples

Table C.7 Autoencoding 50-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Rising-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	16	17	17	15
Decoder size (KB)	75	86	86	39
Avg % error 0.8VDD	2.0579	2.0365	2.0015	2.1108
Avg % error 0.5VDD	2.0528	2.0463	1.9264	2.0596
Avg % error 0.2VDD	2.1393	2.0543	1.9350	2.1574
Mean corcoef	0.99997	0.99996	0.99991	0.99966

Rising-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model Loss (MSE)	2.57E-06	2.71E-06	4.96E-06	3.00E-05
Compression Ratio	3.09	6.13	12.03	24.15

Table C.8 SVD 50-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number

Rising-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.1344	1.3921	0.13929	1.3921	0.17956	2.02019	0.37226	4.305
Avg % error 0.5VDD	0.02621	0.6596	0.03957	0.6596	0.08648	1.00776	0.3875	3.7750
Avg % error 0.2VDD	0.04936	1.0029	0.09402	1.4019	0.20546	3.5549	0.44435	5.9104
Mean corcoef	0.99999		0.99999		0.9937		0.99323	
Model Loss (MSE)	3.27E-8		1.379E-6		1.366E-5		1.416E-4	
Compression Ratio	1.56		3.125		6.25		12.5	

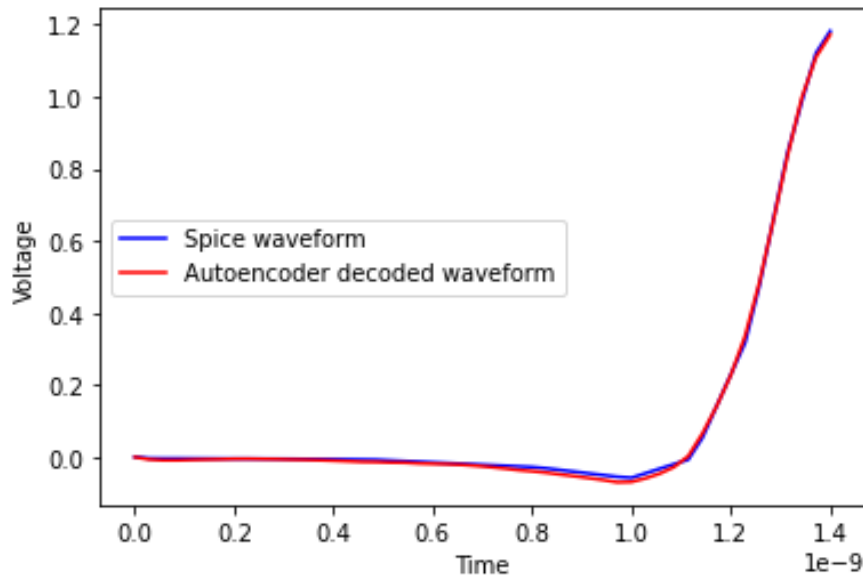


FIGURE C-3 SPICE vs reconstructed 50-points sampled rising edge waveform

C.2.3 Falling-edge 100-samples

Table C.9 Autoencoding 100-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	16	14	17	14
Decoder size (KB)	100	33	111	33
Avg % error 0.8VDD	1.0134	1.2019	0.8707	1.3110
Avg % error 0.5VDD	0.9884	0.9656	0.9338	0.8739
Avg % error 0.2VDD	1.0057	1.0476	0.9652	1.0262
Mean corcoef	0.99997	0.99984	0.99994	0.99948
Model Loss (MSE)	2.65E-06	1.28E-05	6.18E-06	6.18E-05

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Compression Ratio	6.18	12.4	23.8	48.58

Table C.10 SVD 100-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number

Falling-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.07631	3.2092	0.0988	1.27973	0.2062	1.8925	0.5987	15.3771
Avg % error 0.5VDD	0.04704	1.2790	0.0551	0.75622	0.12005	1.3618	0.3688	2.6508
Avg % error 0.2VDD	0.04425	0.4861	0.5997	0.86481	0.1426	1.8896	0.46007	10.9180
Mean corcoef	0.99999		0.99993		0.99953		0.99627	
Model Loss (MSE)	2.804E-7		2.15E-6		1.706E-5		1.64E-4	
Compression Ratio	3.125		6.25		12.5		25	

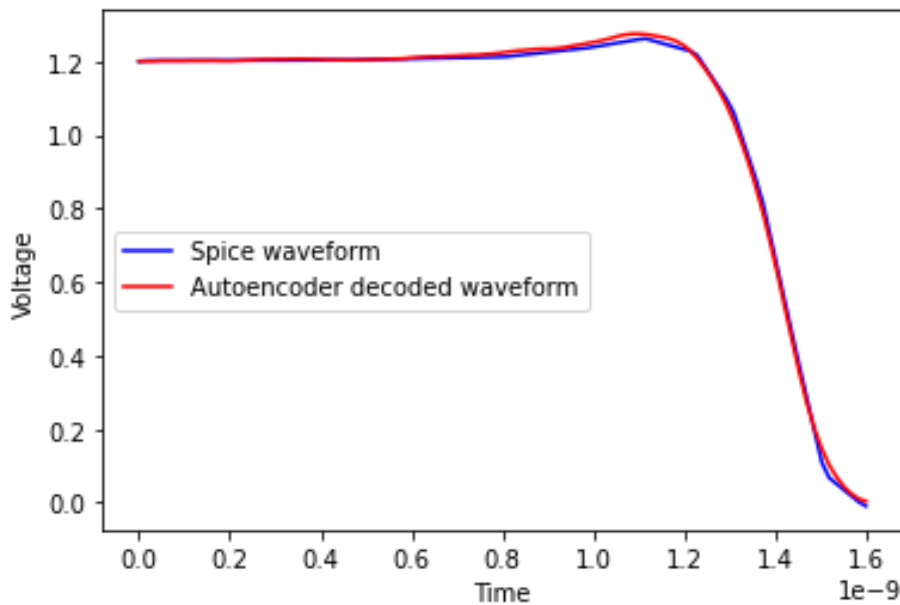


FIGURE C-4 SPICE vs reconstructed 100-points sampled falling edge waveform

C.2.4 Rising-edge 100-samples

Table C.11 Autoencoding 100-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Rising-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	14	15	17	15
Decoder size (KB)	33	51	86	51
Avg % error 0.8VDD	0.9356	0.9321	0.9643	1.1525
Avg % error 0.5VDD	0.9492	0.9436	0.9889	0.9409
Avg % error 0.2VDD	0.9939	0.9538	1.0649	1.2043
Mean corcoef	0.99988	0.99992	0.99992	0.99954
Model Loss (MSE)	8.57E-06	6.21E-06	6.03E-06	5.01E-05
Compression Ratio	6.22	12.35	23.81	47.81

Table C.12 SVD 100-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number

Rising-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.02674	0.3859	0.05112	0.83632	0.11525	1.9355	0.36684	4.0784
Avg % error 0.5VDD	0.01025	0.2249	0.03264	0.44025	0.08862	1.3399	0.4364	4.2364
Avg % error 0.2VDD	0.02754	0.65583	0.07973	1.3464	0.20594	3.6207	0.4608	6.6813
Mean corrcoeff	0.99999		0.99992		0.99945		0.99356	
Model Loss (MSE)	1.878E-7		2.005E-6		1.664E-5		1.746E-4	
Compression Ratio	3.125		6.25		12.5		25	

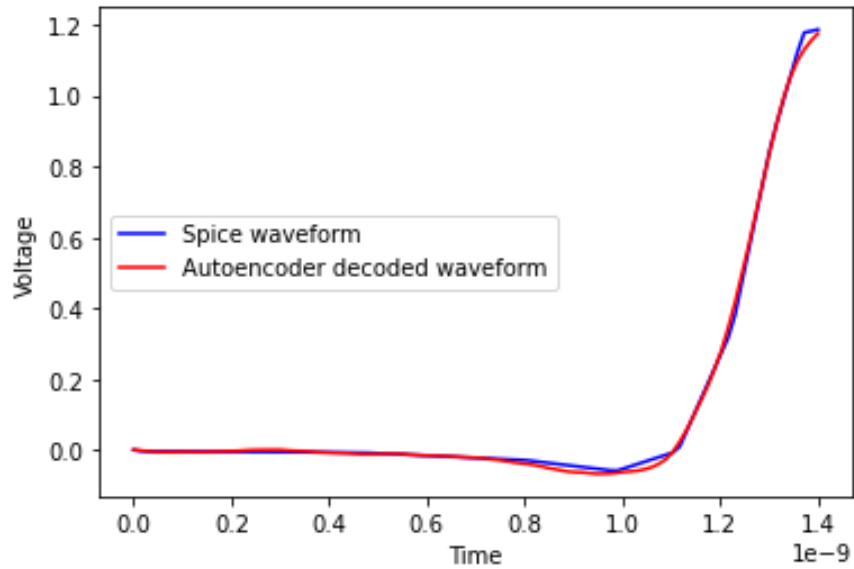


FIGURE C-5 SPICE vs reconstructed 100-points sampled rising edge waveform

C.2.4 Falling-edge 150-samples

Table C.13 Autoencoding 150-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	16	17	16	16
Decoder size (KB)	125	136	125	125
Avg % error 0.8VDD	0.7010	0.5637	0.8729	1.2061
Avg % error 0.5VDD	0.6733	0.5478	0.7194	0.7247
Avg % error 0.2VDD	0.6749	0.6433	0.6641	0.7709
Mean corrcoeff	0.99996	0.99993	0.99992	0.99961
Model Loss (MSE)	3.31E-06	6.59E-06	6.52E-06	4.72E-05
Compression Ratio	9.24	18.19	35.51	67.45

Table C.14 SVD 150-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number

Falling-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.04299	1.01357	0.77262	0.91243	0.19899	1.7279	0.58807	16.449
Avg % error 0.5VDD	0.02473	0.47767	0.03946	0.65335	0.12546	1.7001	0.3929	3.347
Avg % error 0.2VDD	0.02975	0.25947	0.05113	0.69059	0.14689	1.7784	0.4865	15.279
Mean corrcoeff	0.99999		0.99993		0.99953		0.99632	
Model Loss (MSE)	3.0967E-7		2.2919E-6		1.9E-5		1.786E-4	
Compression Ratio	4.6875		9.375		18.75		37.5	

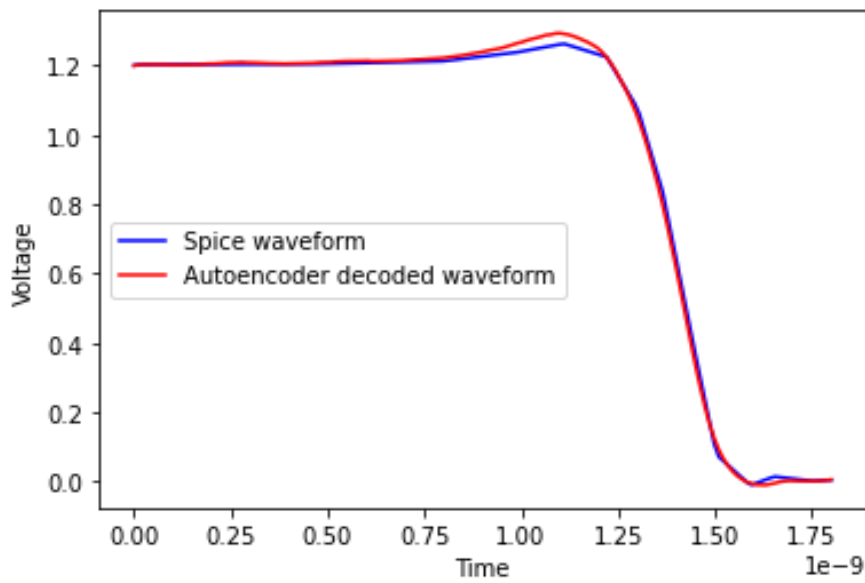


FIGURE C-6 SPICE vs reconstructed 150-points sampled falling edge waveform

C.2.5 Rising-edge 150-samples

Table C.15 Autoencoding 150-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Rising-Edge WF	Autoencoder Number of Encoding Parameters			
	16	8	4	2
Model ID	15	15	16	17
Decoder size (KB)	64	51	125	136
Avg % error 0.8VDD	0.6854	0.7104	0.6484	0.7136
Avg % error 0.5VDD	0.6891	0.6756	0.6564	0.6939
Avg % error 0.2VDD	0.7286	0.7422	0.6800	0.8106
Mean corrcoeff	0.99996	0.99992	0.99993	0.99983
Model Loss (MSE)	3.28E-06	5.89E-06	5.32E-06	1.9E-05
Compression Ratio	9.3	18.48	35.51	66.85

Table C.16 SVD 150-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number

Rising-Edge WF	SVD Sigma Rank Number							
	32		16		8		4	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
Avg % error 0.8VDD	0.02157	0.3424	0.04969	0.76387	0.12358	1.90699	0.41759	4.9855
Avg % error 0.5VDD	0.01063	0.2024	0.03594	0.78928	0.09710	1.69376	0.48040	4.42509
Avg % error 0.2VDD	0.02639	0.67071	0.08528	1.45211	0.21191	3.5906	0.48052	6.33794
Mean corrcoef	0.99998		0.99992		0.99945		0.99317	
Model Loss (MSE)	2.849E-7		2.219E-6		1.926E-5		2.15E-4	
Compression Ratio	4.6875		9.375		18.75		37.5	

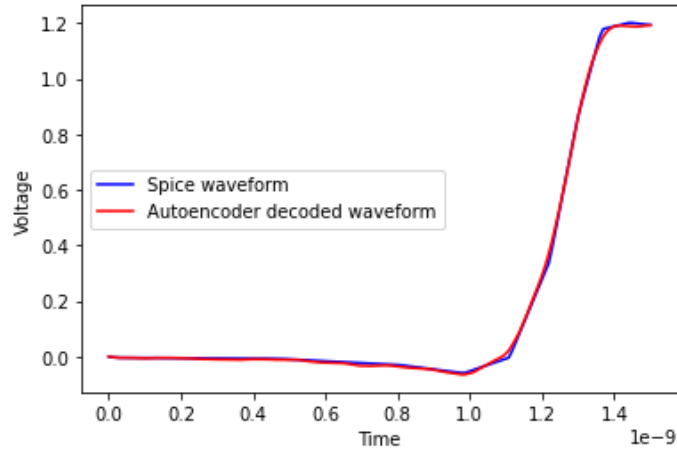


FIGURE C-7 SPICE vs reconstructed 150-points sampled rising edge waveform

C.3 Autoencoding 21-points ECSM Voltage Waveforms

To evaluate the accuracy of the ECSM Autoencoders, we used multiple measures to compare the original waveform with the decoded one. Time-voltage 21-points samples must be normalized to time value between 0 and 1. This is as simple as dividing all the time values by the simulation time used to measure the sample data. Mean correlation coefficient, average percentage error at the key waveform points; 0.2VDD, 0.5VDD and 0.8VDD as well as the overall decoder loss. The best results at different number of encoding parameters are reported in tables C.19 and C.21 for 21-points time-voltage sampled waveforms. Two Autoencoder encoding parameters give less than 1% average percentage error at the key waveform points and average correlation coefficient greater than 0.999 at effective compression ratio of 7.77. Tables C.20 and C.22 shows the SVD compression results at different Sigma rank number. The 2-parameters Autoencoder model gives 1.46x better compression ratio than the nearest SVD compression results at sigma rank equals 4.

Better Autoencoder compression ratio of 15.96 can be achieved at less than 5% average percentage error at key waveform values. On the other hand, SVD runtime decoding is more than 470x faster than Autoencoders decode time because of the small data involved in the SVD analysis, and the large model used for Autoencoders. However, 21-points sampled waveforms don't capture overshoots, undershoots and multiple crossings.

Table C.17 Autoencoding 21-points Falling-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Falling-Edge WF	Autoencoder Number of Encoding Parameters			
	8	4	2	1
Model ID	3	2	4	2
Decoder size (KB)	248	176	391	176
Avg % error 0.8VDD	0.4701	0.4682	0.8755	4.8877
Avg % error 0.5VDD	0.2949	0.2545	0.3878	3.7616
Avg % error 0.2VDD	0.4211	0.4486	0.7502	4.5249
Mean corrcoeff	0.99982	0.99981	0.99911	0.96553
Model Loss (MSE)	1.5E-06	1.4E-06	9.3E-06	4.3E-04
Compression Ratio	2.48	4.86	7.77	15.96

Table C.18 SVD 21-points Falling-Edge ECSM Waveform Results at Different Sigma Rank Number

Falling-Edge WF	SVD Sigma Rank Number			
	8	4	2	1
Avg % error 0.8VDD	0.1984	0.7575	5.0746	8.2802
Avg % error 0.5VDD	0.1168	0.3724	1.4202	5.1151
Avg % error 0.2VDD	0.2273	0.4463	4.5582	6.4531
Mean corrcoeff	0.9997	0.99964	0.98072	0.8738
Model Loss (MSE)	4.20E-07	5.10E-06	2.80E-04	3.6E-03
Compression Ratio	2.62	5.24	10.49	20.99

Table C.19 Autoencoding 21-points Rising-Edge ECSM Waveform Results at Different Number of Encoding Parameters

Rising-Edge WF	Autoencoder Number of Encoding Parameters			
	8	4	2	1
Model ID	2	3	4	2
Decoder size (KB)	176	248	391	176
Avg % error 0.8VDD	0.2979	0.4329	0.4197	1.9021
Avg % error 0.5VDD	0.3665	0.2757	0.3281	3.0917
Avg % error 0.2VDD	0.3616	0.8221	0.8341	6.4361
Mean corrcoeff	0.99991	0.99945	0.99952	0.96948
Model Loss (MSE)	1.40E-06	4.50E-06	4.70E-06	8.6E-04
Compression Ratio	4.86	2.48	7.77	15.96

Table C.20 SVD 21-points Rising-Edge ECSM Waveform Results at Different Sigma Rank Number

Rising-Edge WF	SVD Sigma Rank Number			
	8	4	2	1
Avg % error 0.8VDD	0.1414	0.262	1.5299	1.8536
Avg % error 0.5VDD	0.0948	0.305	0.8058	3.4004
Avg % error 0.2VDD	0.182	0.7886	3.21	6.9688
Mean corrcoeff	0.99998	0.99984	0.99489	0.96318
Model Loss (MSE)	2.25E-07	1.88E-06	5.74E-05	9.8E-04
Compression Ratio	2.62	5.25	10.49	20.99

C.4 NLDM-LUT Results

Table C.21 Falling Edge Mean Percentage Error NLDM-LUT versus Spice

Cell	NOR				INV				NAND			
	100x100		50x50		100x100		50x50		100x100		50x50	
% error	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Delay Time	0.16	18.08	0.41	57.15	0.29	33.13	0.55	57.77	0.24	28.60	0.46	51.55
Tr Time	0.51	52.27	0.97	54.74	0.62	59.75	1.23	59.75	0.46	50.76	0.79	50.76
LUT Size	25x25		7x7		25x25		7x7		25x25		7x7	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Delay Time	0.22	15.46	0.53	15.46	0.37	26.31	0.70	26.31	0.26	23.44	0.61	23.44
Tr Time	0.89	33.49	1.65	33.49	1.42	41.58	2.67	39.85	0.65	38.10	1.32	38.10

Table C.22 Rising Edge Mean Percentage Error NLDM-LUT versus Spice

Cell	NOR				INV				NAND			
	100x100		50x50		100x100		50x50		100x100		50x50	
% error	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Delay Time	0.19	19.04	0.46	55.56	0.40	43.39	0.85	80.48	0.29	27.94	0.67	75.15
Tr Time	0.91	89.17	1.67	89.17	1.01	142.5	2.28	202.7	0.78	54.47	1.62	133.4
LUT Size	25x25		7x7		25x25		7x7		25x25		7x7	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Delay Time	0.34	16.54	0.91	16.54	0.97	31.51	2.82	51.53	0.68	22.97	1.88	27.78
Tr Time	2.12	48.06	4.24	43.85	3.94	153.7	9.25	128.6	2.29	110.5	5.54	57.94

C.5 DL-WFDM results

Table C.23 DL-WFDM Model Training Results using 1000-points Sampled Waveforms

Cell	Falling Edge Waveform Output, Model ID=4						Rising Edge Waveform Output, Model ID=4					
	NOR		INV		NAND		NOR		INV		NAND	
% error	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
@ 0.8VDD	0.61	5.77	0.87	16.38	1.06	14.90	0.59	4.89	1.02	18.13	0.57	7.78
@ 0.5VDD	0.42	5.44	0.72	13.17	0.93	4.95	0.38	2.45	0.54	10.08	0.35	5.46
@ 0.2VDD	0.62	3.48	1.14	17.3	1.21	7.41	0.41	4.20	0.58	4.61	0.41	4.39
Avg Corcoef	0.9995		0.9986		0.9985		0.96435		0.9636		0.9643	
MSE	4.8E-05		1.14E-04		1.41E-04		6.99E-05		1.4E-04		6.72E-05	
Disk size-KB	38						38					
Time - (s)	Training time = 40s - Retrieving time = 0.4s						Training time = 40s - Retrieving time = 0.4s					

Table C.24 DL-WFDM Model Training Results using 150-points Sampled Waveforms

Cell	Falling Edge Waveform Output, Model ID=3						Rising Edge Waveform Output, Model ID=4					
	NOR		INV		NAND		NOR		INV		NAND	
% error	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
@ 0.8VDD	0.82	10.63	1.33	13.94	0.99	9.5	0.87	5.15	0.94	19.31	0.88	9.04
@ 0.5VDD	0.49	6.62	1.05	9.98	0.79	7.93	1.02	6.14	0.94	18.96	0.83	10.08
@ 0.2VDD	0.55	6.41	0.91	8.47	0.66	6.14	1.00	6.06	1.05	20.57	0.93	10.28
Avg Corcoef	0.9962		0.9886		0.9976		0.9943		0.9893		0.9932	
MSE	2.21E-4		5.04E-04		1.69E-04		2.42E-04		4.57E-04		2.9E-04	
Disk size-KB	15						38					
Time - (s)	Training time = 20s - Retrieving time = 0.11s						Training time = 20s - Retrieving time = 0.1s					

APPENDIX D

Published and accepted papers

We have two published papers at one refereed IEEE conference as an outcome of work done in this thesis, and one journal paper is under review at Ain-Shams Engineering Journal.

- **Published:** W. Raslan and Y. Ismail, "Deep Learning Autoencoder-based Compression for Current Source Model Waveforms," *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1-6, doi: 10.1109/ICECS53924.2021.9665573.
 - <https://ieeexplore.ieee.org/document/9665573>
- **Published:** W. Raslan and Y. Ismail, "Structured Recurrent Neural Network Model Order Reduction for SISO and SIMO LTI Systems," *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1-6, doi: 10.1109/ICECS53924.2021.9665593.
 - <https://ieeexplore.ieee.org/document/9665593/>
- **Accepted:** W. Raslan and Y. Ismail, "Deep-learning cell-delay modeling for static timing analysis", *Ain-Shams Engineering Journal*. Manuscript Ref# ASEJ-D-21-01477R1
 - <https://authors.elsevier.com/tracking/article/details.do?aid=101828&jid=ASEJ&surname=Raslan>