Reduced Order Models and Approximations for Hardware Acceleration of Neural Networks

by

Elham Azari

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved May 2021 by the
Graduate Supervisory Committee:

Sarma Vrudhula, Chair
Georgios Fainekos
Fengbo Ren
Yezhou Yang

ARIZONA STATE UNIVERSITY

August 2021

ABSTRACT

Many real-world engineering problems require simulations to evaluate the design objectives and constraints. Often, due to the complexity of the system model, simulations can be prohibitive in terms of computation time. One approach to overcome this issue is to construct a surrogate model, which approximates the original model. The focus of this work is on the data-driven surrogate models, in which empirical approximations of the output are performed given the input parameters. Recently neural networks (NN) have re-emerged as a popular method for constructing data-driven surrogate models. Although, NNs have achieved excellent accuracy and are widely used, they pose their own challenges. This work addresses two common challenges, the need for: (1) hardware acceleration and (2) uncertainty quantification (UQ) in the presence of input variability.

The high demand in the inference phase of deep NNs in cloud servers/edge devices calls for the design of low power custom hardware accelerators. The first part of this work describes the design of an energy-efficient long short-term memory (LSTM) accelerator. The overarching goal is to aggressively reduce the power consumption and area of the LSTM components using approximate computing, and then use architectural level techniques to boost the performance. The proposed design is synthesized and placed and routed as an application-specific integrated circuit (ASIC). The results demonstrate that this accelerator is 1.2X and 3.6X more energy-efficient and area-efficient than the baseline LSTM.

In the second part of this work, a robust framework is developed based on an alternate data-driven surrogate model referred to as polynomial chaos expansion (PCE) for addressing UQ. In contrast to many existing approaches, no assumptions are made on the elements of the function space and UQ is a function of the expansion coefficients. Moreover, the sensitivity of the output with respect to any subset of the input

variables can be computed analytically by post-processing the PCE coefficients. This provides a systematic and incremental method to pruning or changing the order of the model. This framework is evaluated on several real-world applications from different domains and is extended for classification tasks as well.

To my mother and father

for their unconditional love, encouragement and support.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor, Prof. Sarma Vrudhula, for all his guidance and advice throughout the years. Without his persistent help, this dissertation would not have been possible.

I would like to thank my committee members, Prof. Georgios Fainekos, Prof. Fengbo Ren and Prof. Yezhou Yang for reviewing my work, attending my presentations and offering insightful comments.

I would also like to thank the members of the Center for Embedded Systems and other industry members for their technical support. Specifically, I would like to mention Karam Chatha, Rex Hill and Kambiz Yazdi of Qualcomm and Rudy Beraha and Salem Emara of Atlazo for offering me summer internships and fantastic opportunities to learn more about hardware-aware deep learning optimization in practical settings.

Primary funding support for this work was provided by the sponsoring agencies, National Science Foundation- grants #1361926, #1701241 and NSF I/UCRC Center for Embedded Systems. In addition, I would like to thank the school of engineering for awarding me graduate fellowships.

I would like to thank the CIDSE front office staff, including Pamela Dunn and Monica Dugan. A special thanks to Lisa Christian for her help over the past several years.

Finally, and most importantly, I am extremely thankful to my family. My parents and my siblings have been a constant source of encouragement and support throughout these many years. I am forever grateful for everything they have done for me.

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES xii

CHAPTER

1 INTRODUCTION 1

1.1 Surrogate Models 1

1.2 Taxonomy of Surrogate Models 2

1.3 NNs as Data-Driven Surrogate Models 4

1.4 Applications of NN as Surrogate Models 5

1.4.1 Object Detection and Classification 6

1.4.2 Natural Language Processing 9

1.5 Challenges in Employing NN Models 11

1.5.1 Addressing the First Challenge: Need for Hardware Acceleration of NNs 13

1.5.2 Addressing the Second Challenge: Need for Uncertainty Quantification in the Presence of Input Variability 17

1.6 Novelty and Dissertation Structure 18

1.6.1 Addressing the Hardware Acceleration Challenge 18

1.6.2 Addressing the UQ Challenge 19

2 ELSA: A THROUGHPUT OPTIMIZED DESIGN OF AN LSTM ACCELERATOR FOR ENERGY-CONSTRAINED DEVICES 23

2.1 Problem Background: The Need for Hardware Acceleration of LSTM 23

2.2 Existing Hardware Accelerators of LSTM 24

2.3 Problem Statement and Novelty 26

2.4 Background 27

2.4.1 Long Short-Term Memory ... 27

2.5 Architecture of ELSA ... 29

2.5.1 Approximate Multiplier (AM) ... 29

2.5.2 Extension to AM for a Faster Execution ... 31

2.5.3 Comparison with an Exact Multiplier ... 32

2.5.4 Hardware Challenges and Design Decisions ... 34

2.5.5 System Overview ... 35

2.5.6 Main Computation Units ... 35

2.5.7 Controller Units ... 38

2.6 Multi-level Elastic Pipelining ... 40

2.7 Quantization and Accuracy ... 43

2.7.1 Model Description ... 43

2.7.2 Quantized Model ... 44

2.8 Performance Modeling for ELSA ... 48

2.8.1 Pipelined Design ... 49

2.8.2 Non-pipelined Design ... 50

2.9 A Framework for Pre-hardware Mapping Analysis ... 52

2.9.1 Accuracy ... 53

2.9.2 Speed up ... 53

2.9.3 Execution Time ... 54

2.10 Experimental Results ... 54

2.10.1 ASIC Implementation of ELSA ... 54

2.10.2 FPGA Implementation of ELSA ... 61

2.10.3 Summary of the Key Features of This Work ... 64

2.10.4 Opportunities for Further Improvements 65
2.11 Chapter Summary 65
3 AN ALTERNATE DATA-DRIVEN SURROGATE MODEL FOR UNCERTAINTY QUANTIFICATION 67
3.1 Problem Background: The Need for Uncertainty Quantification and Sensitivity Analysis 67
3.2 Challenges in UQ with NN Models 72
3.3 Overview of Polynomial Chaos Expansion 76
3.4 A Data-driven Framework 78
3.4.1 Quantifying Uncertainty with aPC 80
3.4.2 Global Sensitivity Analysis with aPC 81
3.5 Motivation 82
3.6 Incremental Computations to Construct Higher Order Models 87
3.7 Extension to aPC for Classification Tasks 89
3.7.1 Address the Scalability Issue in Classification Tasks 91
3.8 A Computation Graph for aPC 94
3.9 Experimental Results 95
3.9.1 Regression Experiments 95
3.9.2 Classification Experiments 99
3.9.3 Uncertainty Evaluation: Test Examples from Known vs Unknown Classes 101
3.10 Chapter Summary 102
4 USE CASE: POST-FABRICATION WEIGHT TUNING IN A BINARY PERCEPTRON 104

4.1 Problem Background 104
4.2 Problem Statement and Novelty 106
4.3 A Challenging Use Case 107
4.3.1 A Tunable Binary Perceptron 107
4.3.2 Sources of Variations in the FTL Circuit 110
4.3.3 Programming the FTL Circuit 110
4.4 Post Fabrication Weight Tuning 111
4.4.1 The Issues of On-chip PLA 111
4.4.2 Alternate Solution to On-chip PLA 112
4.4.3 Database Construction for FTL Programming 114
4.5 The Proposed Stochastic Simulator 116
4.6 Experimental Results 119
4.6.1 Experimental Setup 119
4.6.2 Results 120
4.7 Chapter Summary 123
5 CONCLUSION 125
REFERENCES 127
APPENDIX
A POLYNOMIAL CHAOS EXPANSION 139
A.1 A Brief History of Polynomial Chaos Theory 140
A.2 Generalized Polynomial Chaos 143
A.2.1 General Polynomial Chaos with Finite Series 145
A.3 Arbitrary Polynomial Chaos 146

LIST OF TABLES

Table Page

1.1 Examples of Surrogate Models. 2

1.2 Taxonomy of Surrogate Models, Adapted from Robinson *et al.* (2008); Asher *et al.* (2015). 3

2.1 Piece-wise Linear Activation Functions Wang *et al.* (2017). 38

2.2 This Shows the Control Flow and the Data Computation in the Proposed Pipelining Method. T Is the Total Number of Time Steps. n Is the Total Number of Hidden Nodes, and j Denotes the j_{th} Component of Its Corresponding Vector. The Output of the Stage Operations as Well as the Mode of Operation for the MVMs Are Specified Below. The Pipeline Stages Shown in Columns Are Executed in Parallel and the Stages Shown in Rows Are Performed Sequentially. 41

2.3 The Relative Error for a Single Multiplication, MAC Operations and an LSTM Layer, as Well as the Classification Accuracy for an Application (i.e. LM That Has Two Consecutive LSTM Layers), When the AM Is Employed in the 8-bit Hardware Design. 47

2.4 The Average Speed-up Achieved by the Pipelining Method over the Non-pipelined Design for *27* Different LSTM Configurations. This Was Computed by Evaluating Equations 2.17 and 2.22. The Minimum and Maximum Speedups Were 1.58x and 1.65x, Respectively. 52

Table Page

2.5 Comparison with the Previous ASIC Implementations. All of These Implementations Are in 65nm Technology. DNPU Is a CNN-RNN Processor, and This Table Only Includes the RNN Values Reported in Shin *et al.* (2017). The LSTM Architecture of DNPU and CHIPMUNK Differ Substantially among Themselves and Also When Compared with ELSA. Moreover, the Reported Applications Are Dramatically Different, Making Comparisons in General Difficult to Judge. 60

2.6 Comparison with Previous Language Modeling Implementations. 62

2.7 Maximum **Improvement** in Throughput and Energy Efficiency as Compared to Prior Implementations. 63

2.8 Resource Utilization of Our Accelerator. 63

2.9 Energy Efficiency of Different Platforms. 64

3.1 RMSE for the Power Plant. 84

3.2 Comparative Results on Regression Benchmarks. 98

3.3 Comparative Results on Image Classification. 101

4.1 An Example of $FabDB$ That Consists of the Failure-types and Their Respective Vt^{Set}. The Order of the Minterms Is [a,b,c,d,e]. For Example, the Failure Type 10000 Corresponds to the Threshold Function a Instead of $ab + ac + ad + ae$. 114

4.2 An Example of $CktFT$ Database for Three Instances of a 5-input FTL and Their Corresponding Failure Types. 116

4.3 The Number of Saved HSPICE Iterations and the Speed up Achieved by Employing the Stochastic Simulator. This Experiment Is Performed on 100 Circuit Instances to Find Their Working Vt^{sets}. 121

Table | Page

A.1 Optimal Polynomials for Various Continuous Distributions Eldred and Burkardt (2012). 144

A.2 Central and Non-central Moments of a Normal Random Variable up to Order 5. 150

LIST OF FIGURES

Figure Page

1.1 (a) An FFNN with Two Hidden Layers. (b) Multiple-input Neuron with an Activation Function f. The Output Is Expressed as $a = f(\sum_{i=1}^{n} x_{\mathrm{i}} w_{\mathrm{i}})$. 6

1.2 Examples of Object Detection and Classification in a Wide Range of Scales and Aspect Ratios Ren *et al.* (2015). 7

1.3 The Architecture of VGG-16 Simonyan and Zisserman (2014) That Consists of 16 Layers With Model Parameters. 8

1.4 The Left Figure Illustrates the Standard RNN. This Network Is Unfolded Over Multiple Time Steps to Demonstrate the Impact of the Vanishing Gradient Problem. As the Input at Time Step One Passes Through the Hidden Layer, it Loses its Sensitivity on the Output, and New Inputs Affect the Hidden Layer. In This Figure, the Degree of Sensitivity is Proportional to the Shades of the Nodes and Gradually Decreases as the Shades Fade Down. 10

1.5 The Structure of an LSTM Layer. It Consists of a Memory Cell (C), an Input Gate (i), an Output Gate (o), and a Forget Gate (f). MVM = Matrix-Vector Multiplier; $\bullet$ = Element-wise Multiplier; σ, tanh = Sigmoid and Hyperbolic Tangent Activation Functions. 11

1.6 Two Distinct Eras of Compute Usage in Training NN Models Amodei and Hernandez (2018). 13

1.7 Server Demand for Deep Learning Inference Across Data centers Park *et al.* (2018b). 14

Figure Page

2.1 The Structure of an LSTM Layer. It Consists of a Memory Cell (C), an Input Gate (i), an Output Gate (o), and a Forget Gate (f). MVM = Matrix Vector Multiplier; • = Element-wise Multiplier; σ, tanh = Sigmoid and Hyperbolic Tangent Activation Functions. 28

2.2 Structure of a 4-bit Signed AM. X and $W \in$ [-1,1) Are the Inputs and Z Is the Product. The Sign bits Are x_3 and w_3. 30

2.3 The Improved Version of the AM. The Modified Parts Are Shown in Red. The Number of States in the FSM Is Reduced by Half and the Down Counter Is Initialized to Half of Its Value as Compared to the One in Figure 2.2. The Preprocessing Unit Sets the Initial Value of the Up-down Counter to 3 (i.e, $\overline{X_3}$ Is One and It Appears Three Times in the Bit-stream in Figure 2.2). Hence, the Initial Value of the Down-counter Is Set to 3, Half of Its Original Value. 31

2.4 The Cell Area Improvement (top) and Power Improvement (bottom) of AM-MAC as Compared to the Exact-MAC for Different Hardware Bit Precision. Each Plot Demonstrates the Accuracy of the AM-MAC for Different Bit Precision as Well. 33

2.5 The Block Diagram of ELSA That Consists of the Computation Units and a Hierarchy of Control Units. 36

2.6 An Example of the MVM Module, in Which It Receives $X_{3\times4}$ and $Y_{4\times1}$ as Its Inputs and Generates the Output Vector $Z_{3\times1}$. 37

2.7 The Controller Hierarchy That Consists of a Top Controller (Top-C) and Three Mini Controllers- MVM-C, EMA-C and EM-C. 39

Figure Page

2.8 The Six Pipeline Stages in the LSTM Layer. $Stage-1$: Eight Parallel MVMs; $Stage-2$: Three Activation Functions and Ternary Adders; $Stage-3$: Two Consecutive Multiplications and an Adder; $Stage-4$: a Sigmoid Function and a Ternary Adder; $Stage-5$: One Tanh Function; $Stage-6$: One Element-wise Multiplication. 42

2.9 The LM Network. It Consists of Two 128 Hidden-node LSTM Layers Followed by FC and Softmax Layers. The Output of Both LSTM Layers Is a Vector of Size 128 and Their Inputs Are of Size 65 and 128, Respectively. . . . 44

2.10 Average MSE for the Hidden State (H) Over 1000 Time Steps for Different Bit-width. The Memory State Demonstrated Similar Behavior, Hence Its Results Were Removed for Brevity. 46

2.11 The MSE (This Work, FP32) of the Hidden State for the 8-bit Quantization. The x-Axis Demonstrates the MSE Values Over 1000 Time Steps. The Dashed Black Line Shows That the Error Does Not Accumulate. The Same Trend Is True for the Memory State for Which Its Results Are Removed for Brevity. 47

2.12 Overall Structure of the Framework. It Consists of Three Main LSTM Implementations in Python That Can Report the Accuracy of This Work for Different Bit-width, the Speed up Achieved with the Pipelining Method and the Execution Time of a Given Application in Clock Cycles. 53

2.13 The Physical Layout of ELSA's Design in 65nm CMOS Technology (Top) and the ASIC's Implementation Results (bottom). 55

2.14 Power (Left) and Area (Right) Breakdown of ELSA's Components Including the SRAMs. AF Stands for Activation Functions. 56

Figure Page

2.15 The ASIC Implementation Results of ELSA as Compared to the Baseline-LSTM. Both of These Designs Were Run at the Same Clock Frequency, the Highest That the Baseline-LSTM Can Achieve. The Reported Numbers Are Normalized. The Energy and Area Efficiency of ELSA Exceeds That of the Baseline-LSTM by Factors of 1.2x. 58

2.16 The ASIC Implementation Results of ELSA as Compared to the Baseline-LSTM. Both of These Designs Were Run at Their Highest Achievable Clock Frequency. The Reported Numbers Are Normalized. The Energy and Area Efficiency of ELSA Exceeds That of the Baseline-LSTM by Factors of 1.2x and 3.6x, Respectively 59

3.1 Different Types of Uncertainty Propagation Methods. 68

3.2 An Example of Bayesian Inference, in Which the Posterior Distribution Is Generated by Multiplying the Prior and Likelihood Distributions. 73

3.3 Implementation of Weight Sampling Method in Deep Learning Models for Diabetic Retinopathy Detection Leibig *et al.* (2017). The Images Are Examples of Fundus Images with Human Label Assignments on Top. Corresponding to Each Image Is the Approximate Predictive Posteriors over the Softmax Output Values $P(diseased|image)$. Predictions Are Based on the Mean of the Posteriors (i.e μ) and Uncertainty Is Quantified By σ. 75

3.4 (a) Sample PE Versus the Predicted PE with aPC for the First 100 Instances. (b) The Frequency Histogram of the Test Error (%) for the 2nd Order Expansion. The Maximum Error is Around 16 Standard Deviations Far From the Mean. 83

Figure Page

3.5 The Bars Represent the Histogram Obtained from the Sample Power Plant Energy and the Curve Shows the Estimated P.D.F of the Energy Using aPC. The KL Divergence Value (d_{Kl}) Between the Two P.D.Fs Is .04, Which Is Close to 0. 85
3.6 Identifying Outliers. The Dashed Line and Band Demonstrate the Estimated Mean with One Standard Deviation. 86
3.7 The Orthogonal Projection of y into W. 88
3.8 The Speed up Achieved by Incrementally Adding to the Computation. This Is Shown up to the $5th$ Order. Greater Speed-up Is Achieved for Larger d. 90
3.9 The Proposed *Training* Procedure for Solving the Scalability Issue and Applying aPC on Handwritten Digit Recognition. The Number of Partitions and the Number of Pixels per Partition Are Denoted as m and n, Respectively. The Output of This Procedure Is m Tuned Coefficients (C) Which Will Be Used for the Inference Phase. 91
3.10 The Proposed *Inference* Procedure for Predicting the Digit Label of a given Test Image Using the Coefficients Computed in the Training Procedure. 92
3.11 The Overall Structure of the Handwritten Digit/Letter Detection Problem for the MNIST Dataset. Typically the Computational Model Is Replaced with LeNet5 as It Has Demonstrated a High Accuracy Of 99.9%. 93
3.12 Computation Graph of aPC for Training. The Inference Graph Has the Same Operations Except the Computation of the Coeffs Node. Hence It Is Omitted for Brevity. 97

Figure Page

3.13 The Normalized Train Time of This Work along with MC Dropout and Deep Ensembles on the Regression Benchmarks. 99
3.14 The Fraction of Train Data Used for Training Deep Ensembles, MC Dropout and This Work on the Regression Benchmarks. 100
3.15 Comparison Between the Execution Time and the Amount of Data Used for Training in This Work and the Other Two Existing Methods For MNIST. 101
3.16 Comparison Between the Execution Time and the Amount of Data Used for Training in This Work and the Other Two Existing Methods For CIFAR-10. 102
3.17 The Percentage of the NotMNIST Images That Are Identified Correctly as Outliers, given Various Intervals. 102
4.1 The Architecture of the FTL Cell with Four Main Components: The Left Input Network (LIN), the Right Input Network (RIN), a Sense Amplifier (SA) and an Output Latch (LA). The LIN and RIN Consist of Two Sets of Inputs $(\ell_1, \cdots, \ell_n)$ and $(r_1, \cdots, r_n)$, Respectively, With Each Input in Series with a Flash Transistor. 109
4.2 Transformation from Boolean Space to Conductivity Space; Hyperplane Gets Converted into a Line. 110
4.3 Proposed Procedure for $FabDB$ Construction. Flow-1 Simulates the Instances Programmed with the Nominal Vt^{Set} and Generates the Failure Types. This is Stored in $CktFT$. The Unique Failure Types are Stored in the First Entry of $FabDB$. Flow-2 Uses a Combination of Circuit Simulation and PLA to Find a Working Vt^{Sets} for the Unique Failure Types Which Is Stored the Second Entry of $FabDB$. 115

Figure Page

4.4 Output Voltage of a 5-input FTL Instance Simulated with the Stochastic Simulator and HSPICE. The Target Function Is [4,1,1,1,1;5]. The Voltages Are Then Thresholded to 0 and 1 to Generate The TT. 118

4.5 Example Distribution of Threshold Functions When the Instances Were Programmed with the Nominal Vt^{Set} For the Target Function [4,1,1,1,1;5]. The Four Most Frequent Functions Are Used for Model Construction. The Model Is Evaluated on the Remaining Circuit Instances. This Strategy Boosts the Accuracy by an Extra 10%. 120

4.6 The Speed up Achieved by Employing the Stochastic Simulator to Reduce the HSPICE Iterations in PLA. Average Speedup of 8.3x Is Achieved with the Maximum Being 56.5X. 121

4.7 The Execution Time (Seconds) of Parallel Circuits Simulated in Both HSPICE and the Stochastic Simulator on the Same Machine. 122

4.8 The Speed up Achieved by Employing the Stochastic Simulator to Reduce the HSPICE Iterations For $FabDB$ Construction. Average Speedup of 6.1x Is Achieved with the Maximum Being 37x. Constructing $FabDB$ Using Only HSPICE Is Impractical Due to the Several Simulations Required to Generate the Failure Types. 123

A.1 An Optimum Projection of Vector $v \in V$ into a Subspace W. 142

Chapter 1

INTRODUCTION

1.1 Surrogate Models

The central problem in the analysis of a physical system is to construct a model of a system denoted by $\mathcal{M} : X \mapsto Y = \mathcal{M}(X)$, in which the input parameters and the output response are represented by vectors $X \in D_X \subset \mathbb{R}^M$ and $Y = \mathcal{M}(X) \in \mathbb{R}^N$, respectively. The model $\mathcal{M}$ may be given implicitly by expressing $\mathcal{M}$ as a solution to one or more differential, integral or algebraic equations. In many cases, $\mathcal{M}$ is a *black box*, in which the model is known through point-wise evaluations, $y^{(i)} = \mathcal{M}(x^{(i)})$ for a given $x^{(i)}$.

Many real-world engineering problems require simulations to evaluate the design objectives and constraints. More often than not, due to the complexity of $\mathcal{M}$, simulations can be prohibitive in terms of computation time. One approach to overcome this issue is to construct *surrogate models*. A *surrogate* model $(\widetilde{M})$ Sudret (2007), also known as *meta-model*, is an approximation of the original computational model $(\mathcal{M})$ that mimics the behavior of model $\mathcal{M}$. This method is employed when model $\mathcal{M}$ is either infeasible or too complex to be constructed. The most common approach to constructing a surrogate model for $\widetilde{\mathcal{M}}$ is to assume a parametric form $\widetilde{\mathcal{M}}(x;\theta)$, where θ is a set of unknown parameters that determine $\widetilde{\mathcal{M}}$. A few examples of surrogate models are presented in Table 1.1. In the next section different types of surrogate models are described.

Table 1.1: Examples of Surrogate Models.

Name of the Model	Form of the Model	Model Parameters θ
Polynomial Chaos Expansions	$\widetilde{M}(x;c) = \sum c_i \Phi_i(x)$	c
Gaussian Process Modeling	$\widetilde{M}(x;\beta,\sigma^2) = \beta^T f(x) + \sigma^2 Z(x,w)$	β, σ^2
Support Vector Machines	$\widetilde{M}(x;w,b) = \sum_{n=1}^{N} w_n K(x_n, x) + b$	w, b
Feed-forward Neural Networks†	$\widetilde{M}(x;w,b) = f^*(\sum_{i=1}^{m} w_j x_j) + b$	w, b

† Function of a single neuron for simplicity.

* f(.) is a nonlinear activation function in the case of classification and is the identity in the case of regression.

1.2 Taxonomy of Surrogate Models

This section describes the taxonomy of surrogate models which is structured based on their mathematical form. The surrogate models are categorized into three different classes, namely, *data-driven surrogates*, *projection-based surrogates* and *multi-fidelity-based surrogates*. Table 1.2, adapted from Robinson *et al.* (2008); Asher *et al.* (2015), presents the taxonomy along with a few examples for each category. The *data-driven surrogates* approximate a complex model based on empirical approximations of the output given the input parameters. *projection-based methods* project the governing equations onto a basis of orthonormal vectors. This leads to reduce the dimensionality of the parameter space. *Multi-fidelity methods* are referred to those that simplify the representation of the physical system by numerical resolution or the underlying physics.

Table 1.2: Taxonomy of Surrogate Models, Adapted from Robinson *et al.* (2008); Asher *et al.* (2015).

Surrogate Category	Also Known As	Examples
Data-driven surrogates	Response surface Statistical methods Black box methods	Polynomials Neural Networks Gaussian Processes Polynomial chaos expansions Bayesian networks
Projection-based surrogates	Reduced order method Reduced basis method Model reduction method	Karhunen-Loeve expansion Dynamic mode decomposition Proper orthogonal decomposition
Multi-fidelity-based surrogates	Multi-scale method Hierarchical method Physical-based method	Multi-scale finite element method Variational multi-scale method

The focus of this work is on the data-driven or black-box methods. Some of the examples are Gaussian processes, neural networks and polynomial chaos expansions. For any engineering optimization problem, it is typically not clear at first, which data-driven method is superior to others. Hence, a comprehensive comparison may need to be performed to analyze the superiority of a method to others for a specific problem. In this surrogate category, the most common method for model validation, is to divide the given data into two sets, referred to as training and validation sets. The training set is used to construct the models and the validation set is to validate the data-driven model.

In the broad field of ML, recently artificial neural networks (NN) have re-emerged as a very popular data-driven surrogate model. In the following sections, we describe the NN models as the principal method for constructing data-driven models.

1.3 NNs as Data-Driven Surrogate Models

Machine learning (ML) algorithms are one example of data-driven *surrogate* models. ML encompasses many different approaches to model construction. All of them start by assuming a parametric form for $\widetilde{\mathcal{M}}$. There are two classes of methods for estimating (also referred to as training) the parameters, referred to as *unsupervised* and *supervised learning*. Unsupervised learning aims at detecting hidden patterns in an unlabeled dataset $\{x^{(i)}, i = 1, 2, ..., n\}$. An example of this category is cluster analysis, which is used for exploratory data analysis for grouping data based on a pattern. In this method, the clusters are grouped based on a measure of similarity. On the other hand, supervised learning considers a training data set, $\mathcal{D} = \{(x^{(i)}, y^{(i)}), i = 1, 2, ..., n\}$, where $x^{(i)}$s are the input attributes and $y^{(i)}$s are the output labels. A learning algorithm aims at finding a function $\widetilde{\mathcal{M}} : X \mapsto Y$, where X and Y are the input and output space, respectively. In classification, the output labels $y^{(i)}$ are discrete (e.g. $y^{(i)} \in \{0, 1\}$), in which the objective is to predict the class label of a new point x. In regression, the output labels are continuous, $y^{(i)} \in \mathcal{D}_Y \subset \mathbb{R}$ and the objective is to predict the output value $\hat{y}$ of a new point x by evaluating the constructed model $\widetilde{M}(x; \theta)$. Some of the existing methods for these two classes are neural networks, Gaussian process models and support vector machines. Among the existing methods in ML, neural networks (NN) have presented a major paradigm shift in computing for numerous applications that involve perception, cognition, search and optimization in very large dimensional spaces.

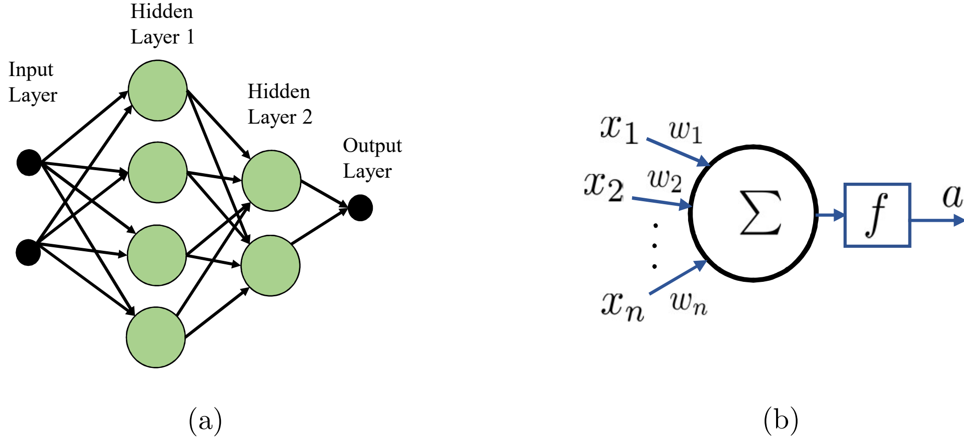
A NN is a data-driven computing model that consists of a group of connected nodes (also known as neurons). NN surrogate models are inspired by the way human brain operates and processes information. Among the many different types of NNs, *feed-forward neural networks (FFNN)* (which are also known as *multi-layer perceptrons*)

are the most commonly used models in nearly every modern application. An FFNN is a directed acyclic graph (DAG) that can be topologically sorted such that the nodes are assigned a level or layer number, with primary input nodes assigned a level 0, and every other node assigned a level that is one more than the maximum level of all its input nodes. In an FFNN, the first and last layers are referred to as the input and output layers, respectively. All the other layers are referred to as *hidden* layers. Figure 1.1a illustrates an example of an FFNN with two hidden layers, where the nodes in the hidden layers correspond to the multi-input neuron depicted in Figure 1.1b. The depth of an FFNN is defined as the number of hidden layers, which is 2 in the example shown in Figure 1.1a. The simplified neuron depicted in Figure 1.1b takes multiple inputs (x_{i}) and produces an output (a). The activation function (i.e. f) can differ from one NN model to another and in some models there is no activation function (e.g. regression). Examples of an activation function is ReLU, sigmoid and Tanh.

In Leshno *et al.* (1993); Lu *et al.* (2017), it was shown that an FFNN with one hidden layer can approximate any continuous function arbitrarily closely if the hidden layer has sufficiently large number of neurons. Convolutional neural networks (CNN) and recurrent neural networks (RNN), which are two other types of NN models, have also been shown to be universal approximators Zhou (2020); Schäfer and Zimmermann (2006). These models are explained in details in the next section.

1.4 Applications of NN as Surrogate Models

Nowadays, NNs are being used in numerous applications that involve classification, text analysis, search and optimization over very large dimensional spaces. Deep Neural Networks (DNN) have achieved excellent results in accuracy and performance for perception, cognition and text understanding. For image recognition, DNNs have



(a) (b)

Figure 1.1: (a) An FFNN with Two Hidden Layers. (b) Multiple-input Neuron with an Activation Function f. The Output Is Expressed as $a = f(\sum_{i=1}^{n} x_{\mathrm{i}} w_{\mathrm{i}})$.

been able to achieve error rates of $< 5\%$ on millions of images from a large and database called ImageNet. This performance equals or exceeds that of humans, and continues to improve. Equally remarkable results have been demonstrated in numerous other application domains, that involve pattern recognition, search and optimization. A few examples are retail, robotics, autonomous driving and health-care. DNNs have defeated the world champion in the games of GO and Poker, have been used to detect viruses in software, to conduct transactions on stock markets, and optimize inventory and supply chain Brynjolfsson and Mcafee (2017).

The two main categories that the existing NN models have been used are (1) *object detection and classification* and (2) *natural language processing.*

1.4.1 Object Detection and Classification

These class of problems are mainly solved by convolutional neural networks (CNNs). CNNs are one type of data-driven surrogate models that can be expressed as the general form $\widetilde{\mathcal{M}}(x;\theta)$, where x is the input and θ is the set of model parameters referred to as *weights*. The input to a CNN (i.e. x) is an image and the output is the class

label that the image belongs to. This is referred to as *classification*. In *object detection*, the objective is to detect instances of the semantic objects of a certain class by delineating their boundaries in a given image or video. Figure 1.2 illustrates the outputs of a CNN for object detection.

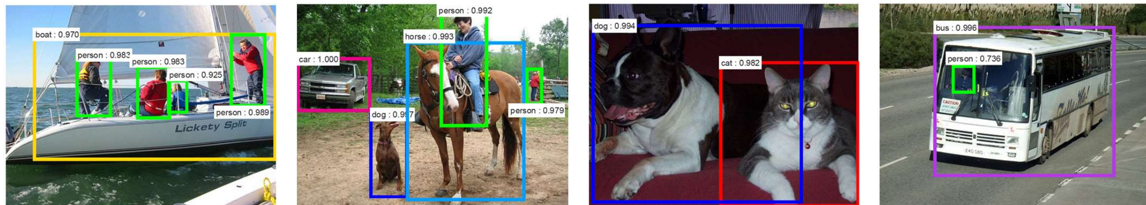


Figure 1.2: Examples of Object Detection and Classification in a Wide Range of Scales and Aspect Ratios Ren *et al.* (2015).

A CNN typically consists of one input, one output and many hidden layers. The hidden layers involve a sequence of convolutional operations which repeatedly compute inner-products of large dimensional vectors. The other operations in CNNs include activation functions (mainly ReLU), pooling, fully connected and classification layers. Figure 1.3 depicts an example of a CNN architecture, referred to as VGG-16 Simonyan and Zisserman (2014), which consists of 16 layers with model parameters. The first few layers of CNNs extract the high level features of the input and the deeper layers extract a combination of these features which are more complex. In the 2012 ImageNet competition Russakovsky *et al.* (2015), a CNN design, referred to as *Alexnet*, won the competition. Since then, CNNs have become the de facto standard in a wide variety of computer vision tasks. Many successful attempts have been made to improve the quality and accuracy and architecture of *Alexnet*. These were typically achieved by utilizing wider and deeper networks. For example, VGGNet Simonyan and Zisserman (2014) and GoogleNet Szegedy *et al.* (2015) were two of the networks that achieved similar performance in ILSVRC 2014 Russakovsky *et al.*

(2015) while demonstrating significantly improved accuracy of classification tasks as compared to *AlexNet*. These networks have achieved accuracy close to or even better than human-level perception. Typically an increase in the size and the computational cost of CNNs lead to higher accuracy given sufficiently large training data. Different variety of CNN architectures have been designed for image/video classification Karpathy *et al.* (2014); Krizhevsky *et al.* (2012); Lin *et al.* (2013); Simonyan and Zisserman (2014); He *et al.* (2016); Szegedy *et al.* (2015), object and face detection Liu *et al.* (2016); Li *et al.* (2015), human pose estimation Toshev and Szegedy (2014), object tracking Wang and Yeung (2013) and many other computer vision tasks.

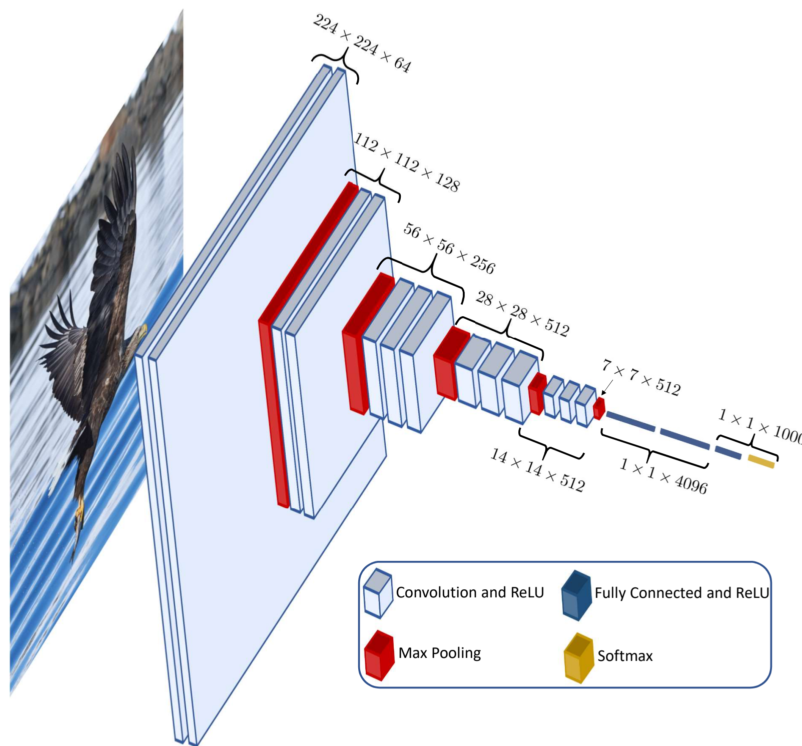


Figure 1.3: The Architecture of VGG-16 Simonyan and Zisserman (2014) That Consists of 16 Layers With Model Parameters.

1.4.2 Natural Language Processing

The data in this specific class of problems are *sequential*, i.e. the individual inputs are *sequences* that have *long-term* temporal dependencies. Some well known examples of problems that have temporal dependencies are handwriting recognition and generation, language modeling and machine translation, speech recognition, prediction of protein structure, audio and video data analysis. In general, data that have temporal dependencies can be processed by recurrent neural networks (RNNs), which unlike CNNs, have feedbacks and are not acyclic graphs. However, standard RNNs have a major drawback that limits their use in a wider range of sequence labeling applications, namely, their inability to accommodate long-term dependencies. This problem arises in situations where the gap between the required information and the place where it is needed is very large. For example, for predicting an upcoming word in a text, where the prediction is dependent on many preceding words, standard RNNs cannot learn to use information so far in the past, which is due to the *vanishing gradient* problem Hochreiter *et al.* (2001). This problem occurs during training RNN and is due to a recurrent weight and derivative of activation function that could be less than one. Figure 1.4 illustrates the general structure of an RNN and the impact of vanishing gradient problem on the output.

To alleviate this problem and be able to learn the patterns over a larger number of time steps, more advanced RNN models, such as Gated Recurrent Unit (GRU) and Long-Short Term Memory (LSTM) have been developed. An LSTM Graves and Schmidhuber (2005); K. Greff et. al. (2017) is a special type of RNN designed to handle the problem of long-term dependencies more accurately than the standard RNNs. This is achieved by the gating mechanism in LSTMs. Figure 2.1 depicts the structure of an LSTM layer.

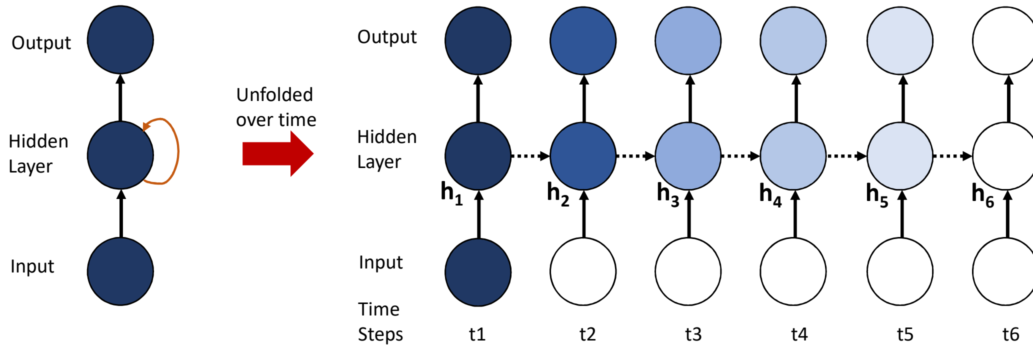


Figure 1.4: The Left Figure Illustrates the Standard RNN. This Network Is Unfolded Over Multiple Time Steps to Demonstrate the Impact of the Vanishing Gradient Problem. As the Input at Time Step One Passes Through the Hidden Layer, it Loses its Sensitivity on the Output, and New Inputs Affect the Hidden Layer. In This Figure, the Degree of Sensitivity is Proportional to the Shades of the Nodes and Gradually Decreases as the Shades Fade Down.

In general, one of the limiting factors in RNNs is the recurrency and the sequential nature. This prevents parallelization during training and batching across examples due to memory constraints. Recently, more advanced networks have been developed based on attention mechanism that eliminate the recurrency in RNNs. Attention-based models take into account several other inputs at the same time and determine which ones are important by assigning different weights to those inputs. Unlike RNNs that take the entire sentence as the input, these models predict based on only a part of the input which has the most relevant information. These networks model the global dependencies instead of the distances between their distance in the input or output sequences. Examples of such networks are Transformer Vaswani *et al.* (2017), BERT Devlin *et al.* (2018), XLNET Yang *et al.* (2019) and RoBERTa Liu *et al.* (2019). These networks are used in machine translation, question answering, sentiment analysis and many other tasks. This is still an active area of research and

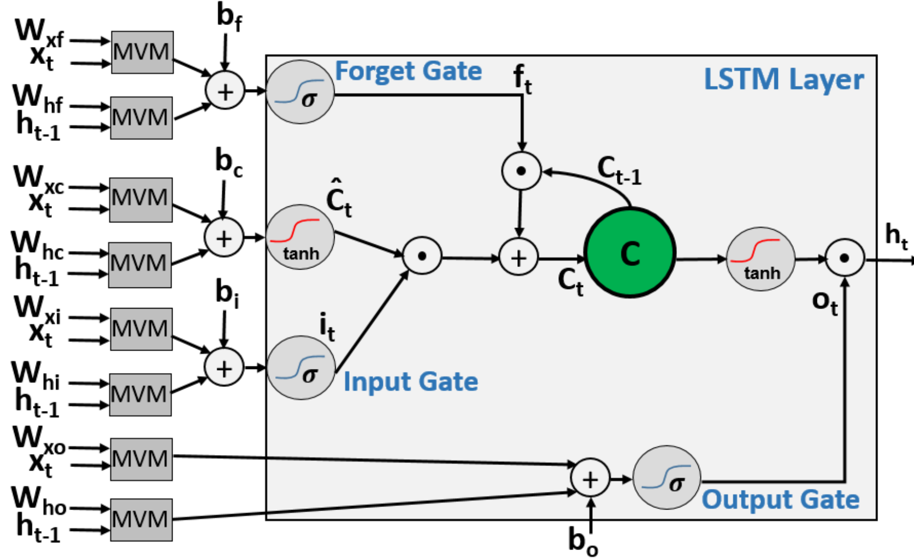


Figure 1.5: The Structure of an LSTM Layer. It Consists of a Memory Cell (C), an Input Gate (i), an Output Gate (o), and a Forget Gate (f). MVM = Matrix-Vector Multiplier; $\bullet$ = Element-wise Multiplier; σ, tanh = Sigmoid and Hyperbolic Tangent Activation Functions.

the accuracy and size of the networks are being improved.

1.5 Challenges in Employing NN Models

Although NN models have achieved excellent accuracy and are used in a wide variety of applications, employing these models pose their own challenges. This section explains two common challenges in details and the existing techniques that address these issues. In the next section, our approaches to address these problems are proposed.

The first challenge in deep NN models is that they involve compute-intensive operations. There are two main phases in NN models, which are referred to as training and inference. After the design of the NN architecture (i.e. deciding on the number of layers and type of NN), the training phase determines the model parameters. During the inference phase, the model is used to predict the output. Typically, GPUs are

the main platforms for the training due to their high computation power in parallel computation. Although the training is a costly procedure in terms of computation time and power consumption, it is typically performed once. However, the inference phase is executed numerous times. The high demand in the inference phase of deep NN models, referred to as deep learning (DL), is due to the ubiquitous use of these models in an expanding range of applications and the continuous improvements in their quality. These often lead to an increase in the computational cost and the memory requirements. Figure 1.6 illustrates the two distinct eras of training NN models in terms of compute usage. It presents the total amount of compute (petaflop/s-days) used to train the models. Since 2012, the compute usage in the largest trainings has been exponentially increasing with a 3.4-month doubling time as opposed to Moore's law that reported a 2-year doubling time.

The second challenge in NN models is when the input space belongs to a stochastic domain. Consider a data-driven surrogate model defined as, $x \in \mathcal{D}_x \mapsto y = \widetilde{\mathcal{M}}(x;\theta)$, where x and θ are the input and parameters of a model, respectively. In many real-world examples, the inputs belong to a stochastic domain and there are uncertainties in the input. Various sources of uncertainty exist that could arise due to the errors in the input data measurements, model structure, parameter values, etc. Let $f_X(x)$ denote the probability density function (p.d.f) of a random variable x. The objective of uncertainty propagation from the input is to estimate the statistics of the output $Y = \mathcal{M}(X)$. Typically NN models only provide point estimates of the parameters and predictions. Quantifying the uncertainty in a prediction has several practical uses: handing over the control to a human or transitioning to a safe mode by an autonomous vehicle or a robot, suggesting further analysis in medical diagnosis, etc.

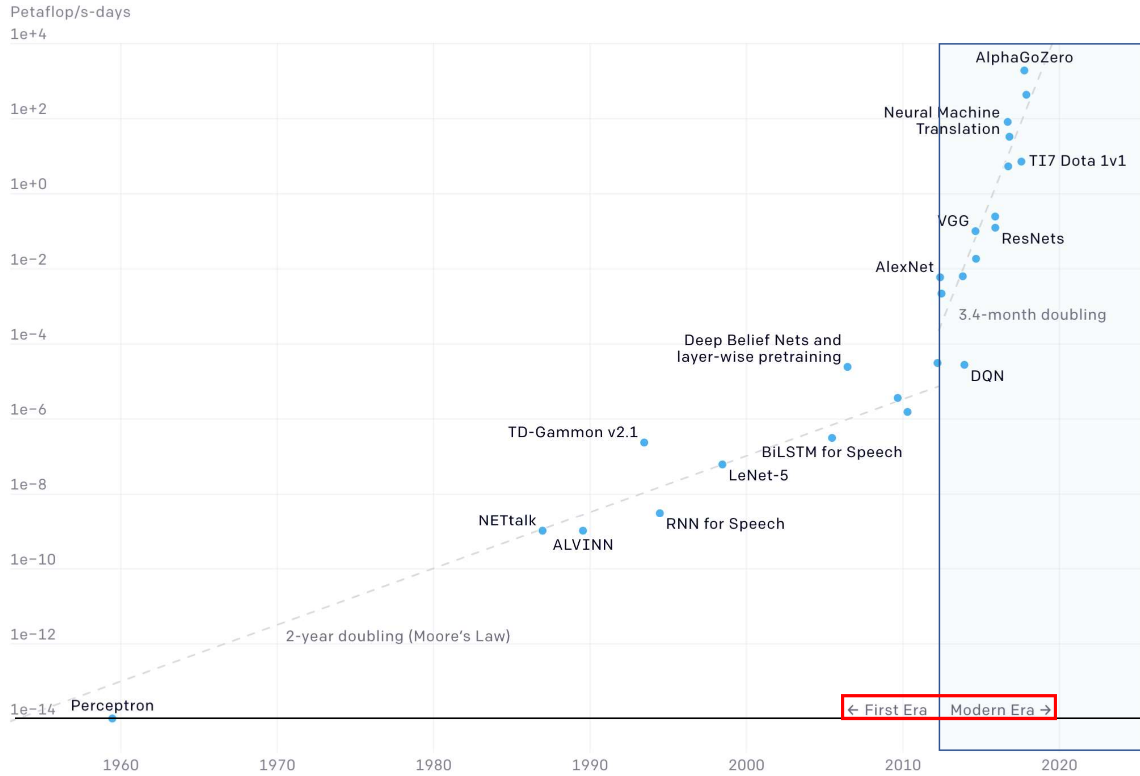


Figure 1.6: Two Distinct Eras of Compute Usage in Training NN Models Amodei and Hernandez (2018).

1.5.1 Addressing the First Challenge: Need for Hardware Acceleration of NNs

Cloud Servers: Norman P. Jouppi et al. (2017) reported in 2013 that people used voice search three minutes a day using speech recognition. This would have required the Google data centers to double to meet computation demands which would have been costly with conventional CPUs. This led to their first generation of custom hardware design, referred to as tensor processing unit (TPU). The TPU was deployed by Google in 2015 and incorporated into their data centers to accelerate the inference phase of NN models. The second generation of TPU was designed and announced in 2017 to improve the memory bandwidth of the first generation TPU. Later in 2018, the third generation of TPU was deployed that was equipped with faster processors.

Google has employed TPUs for many tasks. For example, in Google street view text processing, all the text in the database was found in less than five days, and in Google photos, a single TPU processes over 100 million photos a day. Google stated that TPUs deliver an order of magnitude higher performance per watt than all commercially available GPUs and FPGAs Mah Ung (2018).

Deep learning is also used in many social network services. According to Facebook, their social network services Hazelwood *et al.* (2018) must deliver high quality visual, speech, and language models to billions of users. In 2018, they stated that a significant fraction of the future demand is expected to come from workloads corresponding to inference. Figure 1.7 depicts the server demand from late 2018 up to mid 2020. As a result of this trend, the power consumption in the data centers has been rapidly increasing over time Hazelwood *et al.* (2018).

Cloud servers equipped with traditional general purpose processors and GPUs alone cannot accommodate billions of inferences every day. Moreover, transistor scaling has also slowed down and there is a need to design efficient custom hardware accelerators specialized for the deep learning tasks to significantly improve the performance, power and area of the cloud servers.

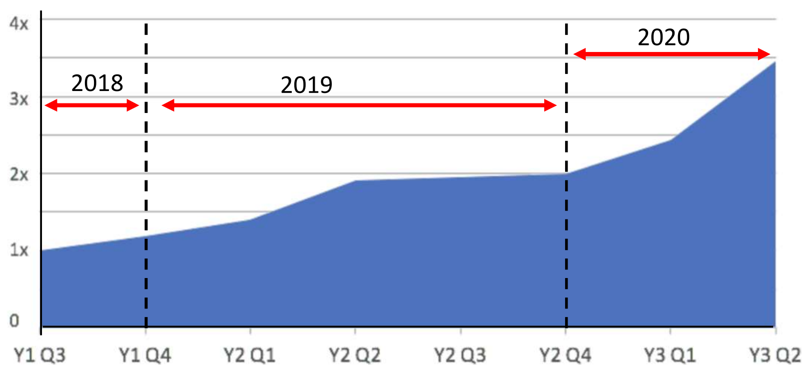


Figure 1.7: Server Demand for Deep Learning Inference Across Data centers Park *et al.* (2018b).

Edge Devices: Computing at the edge brings cloud servers capabilities (in a smaller scale) *closer* to the users and the devices that need them. Edge computing performs the computation near the source of data instead of relying completely on the server. This leads to improving the response time and saving the communication bandwidth. Edge devices are the end-points of a system and today they dominate the personal computing market. Examples of these devices are mobile phones, smart speakers, AR/VR headsets, automotive, security cameras and drones. Although cloud servers provide higher amount of computing power, not all the deep learning tasks should be performed on the cloud servers. Processing at the edge complements the cloud and is a necessity to reducing the load on the servers and bringing higher reliability, faster execution, etc. This is due to the reason that a custom accelerator for deep learning enables the edge device to perform the tasks much faster while consuming less power. This enhances security and privacy and provides reliability, low latency and efficient use of network bandwidth as there is no need to wait for the data to be sent to the cloud, processed and sent back to the device. Thus, it stands to reason that the next significant step in the evolution and proliferation of ML technology will be its integration with edge devices.

Performing the compute-intensive NN models on the edge pose their own challenges. Deep learning workloads are large and compute intensive and are complicated models with complex concurrencies that have to be executed in real-time. The performance of these edge devices are severely limited in their energy capacity and subject to severe thermal constraints. In addition, edge devices are storage and memory limited. Hence the only real choice for ML on edge devices is low power custom accelerator whose architecture is highly optimized for a specific class of ML tasks.

Existing Techniques for Executing NN on Hardware Platforms

Today's NN models involve many layers and nodes. On the other hand, hardware platforms are limited in terms of memory storage, computation resources and are energy constrained. Hence, reducing the models storage and computational cost is essential, and there needs to be holistic system-level solutions for mapping these models onto specialized hardware platforms. Recent years witnessed significant progress in the developing techniques for efficient execution of NN models on hardware platforms both in terms of performance and power. Two common techniques that are used to reduce memory usage and energy consumption and accelerate computation are (1) *Compression* and (2) *quantization.*

Compression Parameter pruning and sharing refers to eliminating parameters which have a small impact on the accuracy of the network. Some of the main techniques are *structured pruning* Anwar *et al.* (2017) and *unstructured pruning* Han *et al.* (2016). In unstructured pruning, the compressed network consists of irregular network connections while the structured pruning results in structured sparsity that is more advantageous in terms of parallel computation and direct mapping to hardware accelerators.

Another category of compression methods is *Knowledge distillation* Hinton *et al.* (2015); Bucila *et al.* (2006). Many of the existing models have billions of parameters and are intended for complex tasks. Deploying such large and complicated models on edge devices is often infeasible. In knowledge distillation, a small network is trained with the help of the larger and more complicated network. After training, the distilled network is able to produce comparable results and in some cases even reproduce the output of the larger network.

Quantization Another technique for reducing the bit-width of the model parameters is quantization. In this approach, the precision of the NN model parameters and/or the activations are reduced from 32-bit floating-point (FP32) to lower bit precision. This technique can substantially reduce the storage requirement, power consumption and custom chip area and hence improve the throughput. However, aggressive quantization typically leads to severe accuracy loss. This is often compensated by retraining, mixed-precision or non-uniform quantization Choukroun *et al.* (2019); Park *et al.* (2018a).

1.5.2 Addressing the Second Challenge: Need for Uncertainty Quantification in the Presence of Input Variability

Existing Techniques for Including Uncertainty Quantification in NNs

Neural networks, particularly deep NN (DNN) models are increasingly being deployed for use in autonomous vehicles, medical diagnosis, robotics, and other safety critical applications. However, the reported successes hide a severe threat lack of an accurate measure of uncertainty associated with the prediction. Knowing the uncertainty in a prediction allows for the handing over control to a human or transitioning to a safe mode by an autonomous vehicle or a robot or suggesting further analysis in medical diagnosis.

Existing methods of uncertainty quantification (UQ) are Bayesian methods Ghahramani (2015) and weight space sampling such as dropout Gal and Ghahramani (2016). Bayesian probability theory provides mathematically grounded tools to reason about model uncertainty, but these usually come with a prohibitive computational cost. In weight sampling method, it was shown that the use of dropout (and its variants) in NNs can be interpreted as a Bayesian approximation of a well known probabilistic

model: the Gaussian process (GP). It was shown that an NN with arbitrary depth and non-linearities, with dropout applied before every weight layer, is mathematically equivalent to an approximation to the probabilistic deep Gaussian process Damianou and Lawrence (2012).

Both of these methods incur very high computational cost and are not very accurate. Another significant drawback of DNNs is the difficulty in determining the sensitivity of the response to any subset of the inputs. This often results in having redundant nodes in the networks, increasing the computational cost and response time, and reducing the flexibility to deploy these networks for real-time and safety-critical applications.

1.6 Novelty and Dissertation Structure

1.6.1 Addressing the Hardware Acceleration Challenge

The first part of this work describes the design of an **e**nergy-efficient **LS**TM **a**ccelerator, referred to as ELSA Azari and Vrudhula (2020). The overarching goal of this work is to aggressively reduce the power consumption and area of the LSTM components using approximation computing, and then use architectural level techniques to boost the performance. This is achieved by two main steps. First, we design and employ low power and compact computation units for the LSTM. Some of these modules use approximate calculations, which require much lower power but incur a high execution time penalty, i.e. it may take multiple clock cycles to finish one operation. Moreover, many of these modules are on the critical path which further degrade the performance. Second, to recover the throughput loss and achieve higher energy efficiency, we develop efficient scheduling techniques that include overlapping of the computations at multiple levels – from the lowest level modules up to the application.

The main results of this work are summarized below.

- The performance of a low power approximate multiplier (AM) is significantly improved and incorporated in the compute-intensive units of ELSA. The execution time of the AM is data-dependent and the number of clock cycles required to finish a single multiplication depends on the magnitude of the multiplicand. An intricate hierarchical control with four distinct, interacting controllers are designed to efficiently synchronize the single-cycle and variable-cycle operations in ELSA.

- To maximize the throughput and compensate for the performance loss, elastic pipeline stages are incorporated at *three* levels. The first one is at the MAC level as these units are *internally pipelined* simultaneously. The second and third levels are at the LSTM layer (overlapping the operations at different time steps) and application, respectively.

- A general performance model of ELSA as a function of hidden nodes and time steps is also presented. This is to permit accurate evaluation of ELSA for any application that includes a network of LSTM layers, such as speech recognition and image captioning.

- The performance and energy efficiency of the LSTM design are further improved by applying post-training, range-based linear quantization to the model parameters.

1.6.2 Addressing the UQ Challenge

The second part of this work aims to develop a robust framework for constructing data driven models using supervised learning. The surrogate model that this work

employs in the presence of input variability is referred to as *arbitrary polynomial chaos expansion (aPC)* which is a data-driven model based on an extension to *polynomial chaos expansion (PCE)*. This is an alternate approach to ML regression methods and is a new approach to training and inference in ML. Consider the model of a physical system $\mathcal{M}$ as explained in Section 1.1, in which the input parameters and the output response are represented in vectors $x \in D_x \subset \mathbb{R}^M$ and $y = \mathcal{M}(x) \in \mathbb{R}^N$, respectively. The problem is to estimate or learn a function that best explains the given set of input-output pairs. This problem is viewed as selecting one function from among an infinite, uncountable space of continuous functions. However, unlike all existing approaches, the proposed approach does not impose a probabilistic model on the function space. Instead, it represents the elements of the function space as multivariate orthogonal polynomials in the underlying input variables. In such a representation, known as *aPC*, the polynomials are constructed directly from the moments of the inputs. These polynomials serve as the basis functions for representing the elements of the function space. In contrast to many existing approaches, no assumptions (e.g., Gaussian processes or prior distributions on the parameters, etc) are made on the elements of the function space. The proposed method also provides a direct method to compute the sensitivity of the model output with respect to any subset of input variables, providing a systematic and incremental method to pruning the model or changing the order of the model. The key advantages of this method are summarized below.

- In addition to point-wise prediction, this approach provides as estimate of the first two moments of the output. Particularly, uncertainty in this method is a simple function of the expansion coefficients of the model, which depend on the raw data. As in other methods, the mean here is also used as the point estimate, and sigma is considered as an estimation to statistical uncertainty.

- Higher order moments of the output and its p.d.f can be estimated using Monte Carlo (MC) methods efficiently. This can be done by evaluating the constructed aPC model for sufficient samples of input. In this approach, computation of the p.d.f is very efficient as evaluating the aPC method is not compute-intensive.

- Global sensitivity analysis can also be used to perform inference in using low accuracy models first, and then switching to higher accuracy models as the need arises. This approach provides a direct method to compute the sensitivity of the model output with respect any subset of input variables, providing a systematic and incremental method to pruning the model or changing the order of the model. In other words, this model allows for efficient GSA for constructing robust models, on the fly, allowing real-time training and inference.

- There is no need to tune hyper-parameters as in NN models and the model can be constructed with limited training data. These are some of the reasons that lead to substantial reduction in computations compared to existing techniques.

The followings outline the topics covered in this dissertation.

- Chapter 2 describes the design of the hardware accelerator in details, including the computation units and the controllers that synchronize them. This work was synthesized and placed and routed in 65nm CMOS technology and it was also prototyped on a Xilinx FPGA Azari and Vrudhula (2019). The results are thoroughly discussed in this chapter as well.

- Chapter 3 explains the aPC method and demonstrates its application on a real-world regression dataset. Then it demonstrates that a constructed model for expansion order d can be used to construct higher order models (e.g. $d+1$). This is shown by incrementally adding to the computation instead of recalculating

everything. This allows a much faster training time. The aPC method is further extended for classification tasks and the scalability issue is also addressed. An executable computation graph is also explained along with the operations that can be parallelized for even faster execution time. The functionality of this approach is then demonstrated on several regression and classification datasets.

- Chapter 4 demonstrates another novel application of aPC – namely, to *tune* parameters of individual instances of manufactured circuits to correct *failures* and consequently maximize yield. While this scenario is quite general, we demonstrate this approach on a recently reported mixed-signal circuit, referred to as *Flash Threshold Logic* (FTL). This circuit performs the function of a *binary neuron* Wagle *et al.* (2019) – i.e., computes a threshold function which is the basic computation involved in binary neural networks Courbariaux and Bengio (2016).

- This dissertation is concluded with a summary of the overall contributions and major findings.

Chapter 2

ELSA: A THROUGHPUT OPTIMIZED DESIGN OF AN LSTM ACCELERATOR FOR ENERGY-CONSTRAINED DEVICES

2.1 Problem Background: The Need for Hardware Acceleration of LSTM

Among the numerous neural network (NN) models, recurrent neural networks (RNN), which are distinguished by the presence of feedback connections, have been shown to be much better suited than feed-forward NNs (e.g. CNN) for many sequence labeling tasks in the field of machine learning (ML) Graves (2012); Sutskever *et al.* (2014). RNNs are designed to capture the temporal dependencies within data sequences, and have been shown to learn the long-term trends and patterns inherent in sequences. To alleviate the *vanishing gradient* problem Hochreiter *et al.* (2001) in standard RNNs, and be able to learn the patterns over a larger number of time steps, more advanced RNN models, such as Gated Recurrent Unit (GRU) Cho *et al.* (2014) and Long-Short Term Memory (LSTM) Hochreiter and Schmidhuber (1997) have been developed. The LSTM model, which is the focus of this work, has been shown to be highly robust and accurate for many applications involving time series data, including natural language processing Dario Amodei et al. (2016) and video analysis Srivastava *et al.* (2015). It is now used in virtual assistant user interfaces such as *Apple Siri*, *Amazon Alexa* and *Google Assistant*. Such applications are typically launched on mobile devices, but due to their compute-intensive nature, they are executed on cloud servers. With the emergence of *Internet of Things* (IoT) and the further proliferation of mobile devices, this approach will not be scalable, and hence there is a need to move some or all of the NN computations to (energy-constrained, performance-

limited) mobile devices. This poses difficult challenges associated with simultaneously achieving high energy-efficiency and high throughput. These challenges are due to the recursive structure of the LSTM model and the compute-intensive operations on very large dimensional data as well as the high memory-bandwidth requirement for computing on a large number of parameters. The goal of this paper is to achieve high energy efficiency by employing low power and compact computation units and aggressively maximizing the overall throughput.

2.2 Existing Hardware Accelerators of LSTM

Implementations of LSTM on CPU-GPU architectures Stollenga *et al.* (2015); Hwang and Sung (2015); Das and Han (2015) are not suitable for mobile devices because of their high power consumption. Although, implementations of LSTM on FPGAs Chang and Culurciello (2017); Li *et al.* (2015); Guan *et al.* (2017b); Lee *et al.* (2016); Han *et al.* (2017); Wang *et al.* (2018); Rybalkin *et al.* (2018); Ferreira and Fonseca (2016); Guan *et al.* (2017a); Rybalkin *et al.* (2017); Nurvitadhi *et al.* (2016); Fowers *et al.* (2018); Cao *et al.* (2019), have been shown to be much more energy-efficient than GPUs, their power consumption is still too high (usually more than 10W) for energy-constrained systems. This has motivated the design of ASICs for LSTMs Norman P. Jouppi et al. (2017); Wang *et al.* (2017); Shin *et al.* (2017); Conti *et al.* (2018).

TPU Norman P. Jouppi et al. (2017) describes a hardware accelerator for inferencing at cloud-scale, specialized for CNNs, Multi-Layer Perceptrons and LSTMs. Despite achieving high throughput (2.8-3.7 TOPS), it consumes up to 40W (not suitable for edge devices) and has low utilization when using LSTMs.

ASIC implementations described in Wang *et al.* (2017); Shin *et al.* (2017); Conti *et al.* (2018) report power consumptions in the tens of milliwatts, while achieving

sufficiently high throughput. Wang *et al.* (2017) present a memory efficient ASIC design for on-line training and classification and demonstrate its functionality for language modeling and speech recognition. To eliminate off-chip memory use, the model parameters are reduced by using circulant matrices along with a compression technique. The reported results are only at synthesis level. DNPU Shin *et al.* (2017) is a reconfigurable CNN-RNN processor with the CNN being its major component and the LSTM as its secondary unit. Quantization and table-based multiplication techniques result in significant reduction in on-chip memory storage. However, this limits its peak performance by requiring the use of external memory even for models with small number of parameters. CHIPMUNK Conti *et al.* (2018) is a scalable LSTM accelerator that is designed to handle applications that operate on large datasets. It achieves significant reduction in the memory transfer overhead by allowing the interconnection of multiple LSTM units in a systolic array structure. However, the power consumption of the *systolic array structure* is too high at an application level and not suitable for energy-constrained edge-devices.

There exists CNN ASIC implementations that employ efficient multipliers for performing the convolution operations Albericio *et al.* (2017); Judd *et al.* (2016). Albericio *et al.* (2017) present a massively data-parallel architecture for CNNs that eliminates most of the ineffectual computations by using a serial-parallel shift-and-add multiplication (Pragmatic unit). The Pragmatic unit as well as its previous bit-serial version (Stripes Judd *et al.* (2016)) are designed to efficiently perform the *inner products* in the convolution layers and also skip the zero bits in the activations. Albericio *et al.* (2017) report 92% zero bits in the activation values of CNNs. One of the reasons is due to the rectified linear (ReLU) activation function that converts negative activations to zero, resulting in many zero activations and no negative activations. Although the multiplier designs in Albericio *et al.* (2017) and Judd *et al.* (2016)

lead to improvements in energy efficiency as compared to its equivalent state-of-the-art CNN accelerators, they cannot be used in the multiplication operations in an LSTM network because of two main reasons. First, there is no convolution operation involved in a typical LSTM network. Second, many of the zero bits in the activations of CNNs as shown in Albericio *et al.* (2017) are due to the ReLU activation function, which does not exist in a typical LSTM network. The sigmoid and tanh functions are used in an LSTM network that have different properties as compared to ReLU.

2.3 Problem Statement and Novelty

Existing ASIC implementations of the LSTM model are based on *conventional* architectures. This paper describes the design of an **e**nergy-efficient **LS**TM **a**ccelerator, referred to as ELSA. The overarching goal of this work is to aggressively reduce the power consumption and area of the LSTM components, and then use architectural level techniques to boost the performance. This is achieved by two main steps. First, we design and employ low power and compact computation units for the LSTM. Some of these modules use approximate calculations, which require much lower power but incur a high execution time penalty, i.e. it may take multiple clock cycles to finish one operation. Moreover, many of these modules are on the critical path which further degrade the performance. Second, to recover the throughput loss and achieve higher energy efficiency, we develop efficient scheduling techniques that include overlapping of the computations at multiple levels – from the lowest level modules up to the application. The main results of this work are summarized below.

1. The performance of a low power approximate multiplier (AM) is significantly improved and incorporated in the compute-intensive units of ELSA. The execution time of the AM is data-dependent and the number of clock cycles required to finish a single multiplication depends on the magnitude of the multiplicand.

An intricate hierarchical control with four distinct, interacting controllers are designed to efficiently synchronize the single-cycle and variable-cycle operations in ELSA.

2. To maximize the throughput and compensate for the performance loss, elastic pipeline stages are incorporated at *three* levels. The first one is at the MAC level as these units are *internally pipelined* simultaneously. The second and third levels are at the LSTM layer (overlapping the operations at different time steps) and application, respectively.

3. A general performance model of ELSA as a function of hidden nodes and time steps is also presented. This is to permit accurate evaluation of ELSA for any application that includes a network of LSTM layers, such as speech recognition and image captioning.

2.4 Background

2.4.1 Long Short-Term Memory

Figure 2.1 shows a typical LSTM layer. The input is a temporal sequence $X = (x_1, x_2, \cdots, x_T)$ and the output is a sequence $h = (h_1, h_2, \cdots, h_T)$, referred to as the *hidden state*, that is generated iteratively over T time steps. The memory cell (C) stores some part of the past history over a specific period of time. At each iteration, the *input gate* controls the fraction of the input data to be remembered and the *forget gate* determines how much of the previous history needs to be deleted from the current memory state (C_t). The output gate decides how much of the processed information needs to be generated as the output (h_t). In a sequence learning task, let $X = (x_1, x_2, \cdots, x_T)$, where x_t is the input to the LSTM layer at time step $t \in [1, 2, ..., T]$. The following equations show how the output sequence $h = (h_1, h_2, \cdots, h_T)$ of a layer

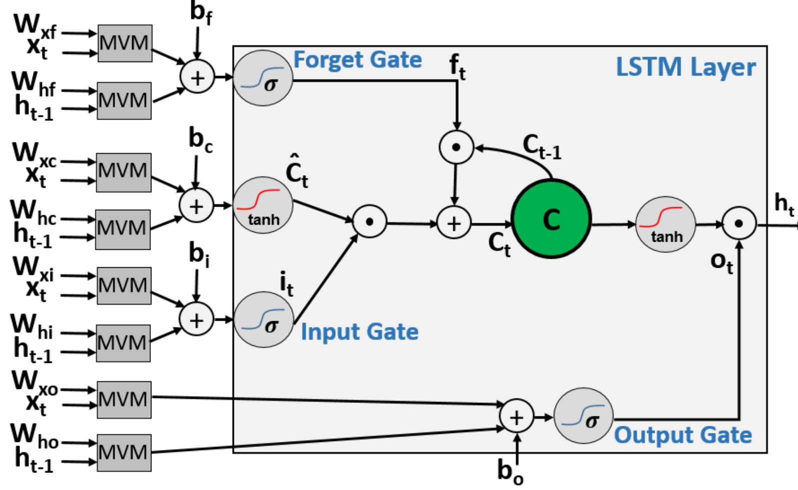


Figure 2.1: The Structure of an LSTM Layer. It Consists of a Memory Cell (C), an Input Gate (i), an Output Gate (o), and a Forget Gate (f). MVM = Matrix Vector Multiplier; • = Element-wise Multiplier; σ, tanh = Sigmoid and Hyperbolic Tangent Activation Functions.

is generated iteratively over T time steps:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), \tag{2.1}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), \tag{2.2}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f), \tag{2.3}$$

$$\hat{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \tag{2.4}$$

$$C_t = i_t \odot \hat{C}_t + f_t \odot C_{t-1}, \tag{2.5}$$

$$h_t = o_t \odot \tanh(C_t). \tag{2.6}$$

The element-wise multiplication is indicated by $\odot$. The parameters are the bias vectors (bs) and the weight matrices (Ws) which are tuned during model training. $\hat{C}_t$ is the new candidate memory which contains the extracted information from the input. The non-linear activation functions, $\sigma \in (0, 1)$ and $tanh \in (-1, 1)$, are defined in Equations 2.7 and 2.8.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.8}$$

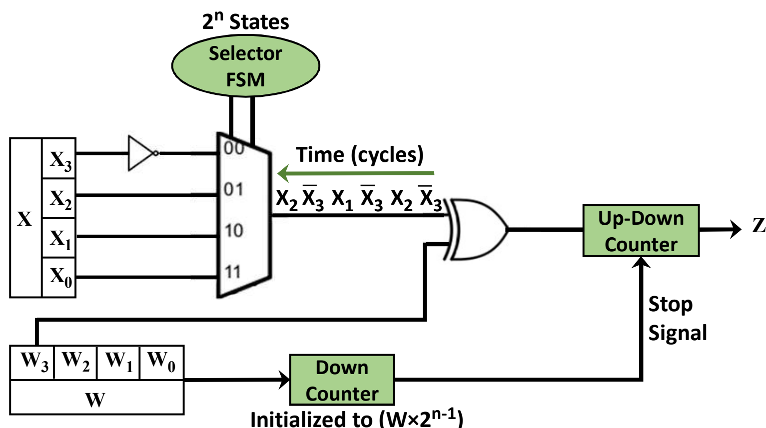
The main challenges in the design of an LSTM architecture is the large number of matrix-vector multiplications (MVMs) involving large dimensional vectors, the element-wise multiplications (EMs) and the data movements from/to the memory.

2.5 Architecture of ELSA

2.5.1 Approximate Multiplier (AM)

The advantages of the approximate multiplier (AM) design in Sim and Lee (2017) are its low logic complexity and reduced power consumption. The AM generates an *approximate* (but sufficiently accurate) product. The inputs and outputs of the AM are represented as signed, fixed-point fractions, i.e., in a binary fraction $X = x_{n-1}.x_{n-2}x_{n-3}...x_0$, the sign bit is x_{n-1} and the fraction is $x_{n-2}x_{n-3}...x_0$. Let $\mathcal{N}$ denote the numerator of a fraction. For example, in a 4 bit number $X = 1.010$, $\mathcal{N}(X) = -6$ and its decimal value is $-6/8$. Let X and $W \in [-1, 1)$ be the inputs of the AM in an n bit multiplication. The exact product is XW. The AM produces an n-bit $Z \approx XW$. The main component of the AM is an FSM (with 2^n states) that generates a specific bit-stream $\{S\}$. The generation of this bit-stream depends on the value of one of the operands, say X and its length is $|\mathcal{N}(W)|$. Specifically, in $\{S\}$, X_{n-i} appears at cycle 2^{i-1}, and then after every 2^i cycles, for $i \in [1, n]$. The main property of $\{S\}$ is that the difference in the number of ones and zeros is an integer approximation to XW. The theoretical upper-bound on the approximation error is $n/2^{n+1}$, but has been empirically shown to be far smaller, approaching the precision of floating point for $n \geq 8$ Sim and Lee (2017). The structure of a 4-bit signed AM along

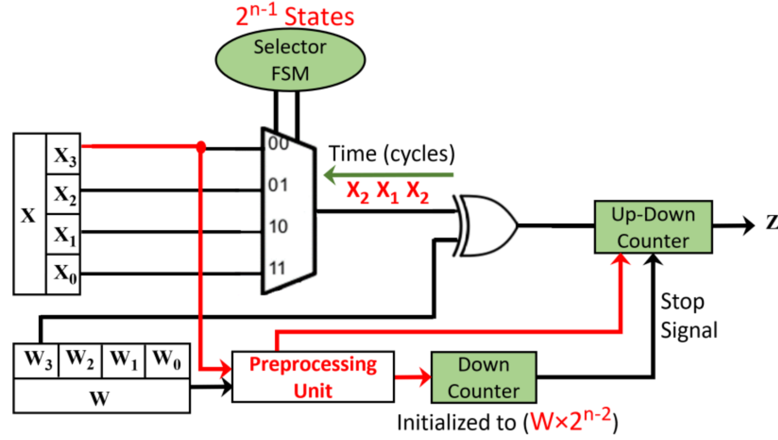
with an example is illustrated in Figure 2.2. In this example, $X = 5/8$, $W = 6/8$ and $n = 4$. The FSM consists of 2^4 states. The FSM-MUX combination generates the bit-stream $S = \{\overline{x_3}\, x_2\, \overline{x_3}\, x_1\, \overline{x_3}\, x_2\}$ at cycles $\{c_1\, c_2\, c_3\, c_4\, c_5\, c_6\}$, respectively. The up-down counter counts up as it receives one and counts down when it receives a zero and generates the product Z. The down-counter stops the AM after $|\mathcal{N}(W)|$ cycles, which is 6 (i.e. $(6/8) \times 2^3$), in this example. The XOR gate receives the bit-stream $\{S\}$ and the MSB of W (w_3), and generates the input of the up-down counter in sequence. Then, the output of the AM is produced by the up-down counter, which is 0.5. The exact result is 0.46, as shown in the table.



X	W	Input of the Up-Down Counter	Output of the AM	Exact Result
$x_3\, x_2\, x_1\, x_0$ 0. 1 0 1	$w_3\, w_2\, w_1\, w_0$ 0. 1 1 0	c_6 c_5 c_4 c_3 c_2 c_1 1 1 0 1 1 1	$\frac{4}{8} = 0.50$	$\frac{30}{64} = 0.46$

Init. Value of the Down Counter	Init. Value of the Up-Down Counter
6	0

Figure 2.2: Structure of a 4-bit Signed AM. X and $W \in$ [-1,1) Are the Inputs and Z Is the Product. The Sign bits Are x_3 and w_3.



X	W	Input of the Up-Down Counter	Output of the AM	Exact Result
$x_3\, x_2\, x_1\, x_0$ 0. 1 0 1	$w_3\, w_2\, w_1\, w_0$ 0. 1 1 0	$c_3\;\; c_2\;\; c_1$ 1 0 1	$\frac{4}{8} = 0.50$	$\frac{30}{64} = 0.46$

Init. Value of the Down Counter	Init. Value of the Up-Down Counter
3	3

Figure 2.3: The Improved Version of the AM. The Modified Parts Are Shown in Red. The Number of States in the FSM Is Reduced by Half and the Down Counter Is Initialized to Half of Its Value as Compared to the One in Figure 2.2. The Preprocessing Unit Sets the Initial Value of the Up-down Counter to 3 (i.e, $\overline{X_3}$ Is One and It Appears Three Times in the Bit-stream in Figure 2.2). Hence, the Initial Value of the Down-counter Is Set to 3, Half of Its Original Value.

2.5.2 Extension to AM for a Faster Execution

The AM is a compact design with low power consumption, however, its execution time is high as compared to the fixed-point exact multiplier. Hence, the design of the original AM is modified to improve its execution time by $2X$ with negligible logic overhead ($< 0.03\%$), as shown in Figure 2.3. This is achieved by adding a small *preprocessing unit* to extract the FSM patterns for the MSB of the first operand and

initializing the up-down counter by the computed value. As the FSM in the original design selects the MSB every two cycles, this modification leads to decreasing the latency of the original AM by 50%. The preprocessing unit consists of an *inverter*, a *shifter* and an *XOR* gate. This unit receives the MSB of X (i.e. $\overline{X_3}$ in this example) and W as its inputs, and generates two outputs as the initial values of the down-counter and the up-down counter. It shifts the value of W to the right by one and sets it as the initial value of the down-counter. The same operation is performed to set the initial value of the up-down counter, except that the sign of the computed value needs to be specified. The sign is determined based on the result of $(\overline{X_3} \oplus W_3)$ which is computed once in the preprocessing unit. If it is a one, the sign is positive, otherwise it is negative. For the example shown in Figure 2.2, the FSM-MUX in this AM generates the bit-stream $S = \{x_2\, x_1\, x_2\}$ at cycles $\{c_1\, c_2\, c_3\}$, respectively. This results in saving 3 (50%) cycles as compared to the one shown in Figure 2.2. The multipliers in ELSA employ this accelerated AM to achieve higher throughput, while maintaining low area and power consumption.

2.5.3 Comparison with an Exact Multiplier

Employing AMs to perform the compute-intensive operations (i.e. MVM) can result in significant savings in both area utilization and power consumption. To explore this, the AM (labeled AM-MAC) and an exact fixed-point multiplier (labeled Exact-MAC) were designed and compared when used in MAC units. Each of these MACs consist of 100 individual multipliers and adders to perform 100 MAC operations in parallel. These units were synthesized using Cadence® GENUS running at 200MHz, for various bit widths ranging from 8 to 16 bits. Figure 2.4 shows the improvement in power and cell area of AM-MAC as compared to the Exact-MAC. Each plot also shows the accuracy of the AM-MAC as compared to the Exact-MAC for various bit

widths. As the bit precision increases, the accuracy of the AM-MAC improves from

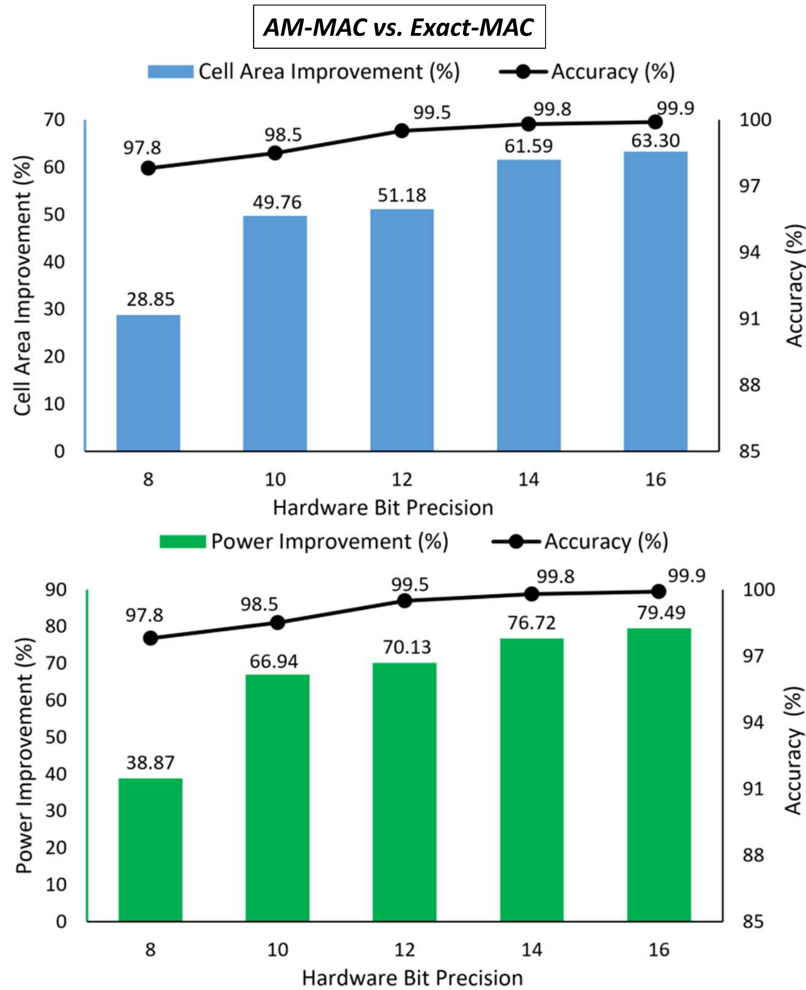


Figure 2.4: The Cell Area Improvement (top) and Power Improvement (bottom) of AM-MAC as Compared to the Exact-MAC for Different Hardware Bit Precision. Each Plot Demonstrates the Accuracy of the AM-MAC for Different Bit Precision as Well.

97.8% to 99.9%, and the maximum savings in the power consumption and cell area reaches 79.49% and 63.30%, respectively. **Note:** Delay comparison of these units in isolation is not meaningful as the AM requires variable number of cycles (i.e. data-dependent) for a single multiplication. Delay comparison of an LSTM network for an application is more meaningful and is described and quantified in Section 2.10.2.

2.5.4 Hardware Challenges and Design Decisions

Although employing AM leads to substantial reduction in area and power consumption, this variable-cycle multiplier poses a number of design challenges. One is the increased latency of the MVM and EM units, both of which lie on the critical path. ELSA's modification of the AM includes elimination of one state of the FSM, and results in improving the AM's performance by 2X. This in itself is not enough. Hence, ELSA's design maximally overlaps the execution of the MVMs with other computation units and over multiple time steps, resulting in a multi-level pipelined design. In addition, the control unit is organized as a two-level hierarchy to efficiently synchronize the AM units and overlap their computations to practically eliminate the waiting time (e.g. arising from being a variable-cycle multiplier) and hide their latency. Finally the potential loss of accuracy due to the presence of feedback and use of AMs is addressed by experimentally evaluating the optimal bit precision for the overall design. The optimal bit precision of ELSA is evaluated by comparing its accuracy with two corresponding LSTM designs. The first one is the software implementation with floating-point calculations, and the second one is an LSTM design with *exact* fixed-point multiplications. This is performed for the following reasons:

1. to explore the impact of using the AMs in ELSA on error propagation through the LSTM for different bit precision and to investigate whether the error accumulates in the hidden and memory states over various time-steps. This is performed by measuring the mean squared error between the hidden states/memory states of ELSA and the floating point implementation.

2. to evaluate the best hardware bit-precision for ELSA that is a good trade-off between its accuracy and its hardware design metrics (i.e. power, area, performance). This is performed by calculating the classification accuracy of ELSA and compar-

ing it with its corresponding exact fixed-point implementation.

2.5.5 System Overview

The top-level block diagram of ELSA is shown in Figure 2.5. It includes the computation units in LSTM as well as the controllers that synchronize them. The computation units include: 1) the MVM modules to perform the matrix-vector multiplications in an LSTM layer in parallel. 2) The ternary adders to perform the addition on the outputs of the MVMs and the bias vectors. 3) the non-linear activation functions, i.e., sigmoid and tanh. 4) EMA module to compute the elements of the memory state. 5) EM module to compute the elements of the hidden state. The control of the AM units is performed by a top controller in coordination with three distinct mini controllers-MVM-C, EM-C and EMA-C. The reason to include mini controllers is that the computation modules that employ the AM units (i.e. MVM, EM, EMA), involve variable-cycle operations and hence require synchronization with other single-cycle operations. Moreover, these units have to execute in parallel to maximize throughput. The required network parameters are loaded into the SRAMs, and the data transfer for fetching/storing the parameters from memory is controlled by the controller units. The intermediate results of the computation units are written into the buffers to reduce the SRAM access time. Thus, the SRAMs are only accessed for fetching the parameters and storing the computed values for the hidden and memory-states. The components of ELSA as well as the multi-level elastic pipelining technique are explained in details in the following sections.

2.5.6 Main Computation Units

MVM Module: The MVM module is a compact combination of the AM units that receives a matrix $X_{n\times m}$ and a vector $Y_{m\times 1}$ as inputs. There are totally n AM units

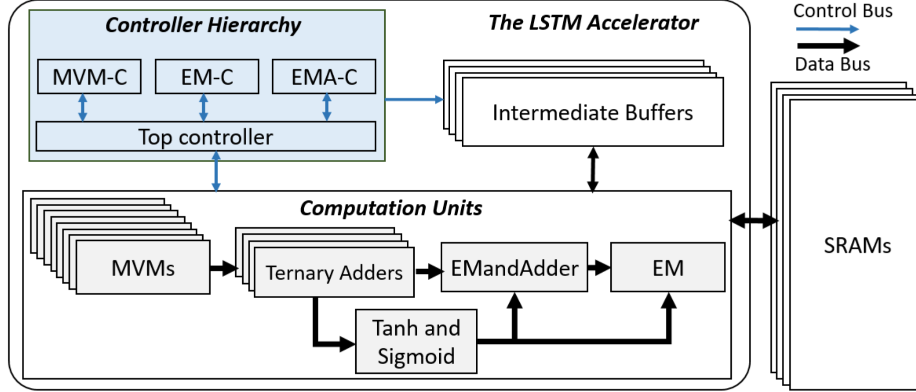


Figure 2.5: The Block Diagram of ELSA That Consists of the Computation Units and a Hierarchy of Control Units.

in an MVM module that all share the same FSM and down-counter, thereby making the module compact. This unit is *internally pipelined* with m pipeline stages. The parallel matrix vector multiplication in the MVM module is performed by multiplying one column of matrix X with one element of vector Y at a time. To store the MAC results, the up-down counter performs as an accumulator and its bit-width is increased by a few bits to preserve the precision. In the example shown in Figure 2.6, at time t_1, the first column-scalar multiplication is performed on column $[x_{11}, x_{21}, x_{31}]^T$ and scalar y_1. The latency of these multiplications which execute in parallel is determined by y_1, and the first partial results are accumulated in the up-down counters. Without reseting the up-down counters, this process is repeated until time t_4, at which the last column-scalar multiplication (i.e. $[x_{14}, x_{24}, x_{34}]^T \times y_4$) is computed and the final output vector $Z_{3\times1}$ is generated. As shown in Figure 2.6, the difference between the start and end times of the operations are not necessarily equal due to their variable cycle execution.

EM and EMA modules: The Element-wise Multiplier (EM) and Element-wise Multiplier and Adder (EMA) modules employ the accelerated AM shown in Sec-

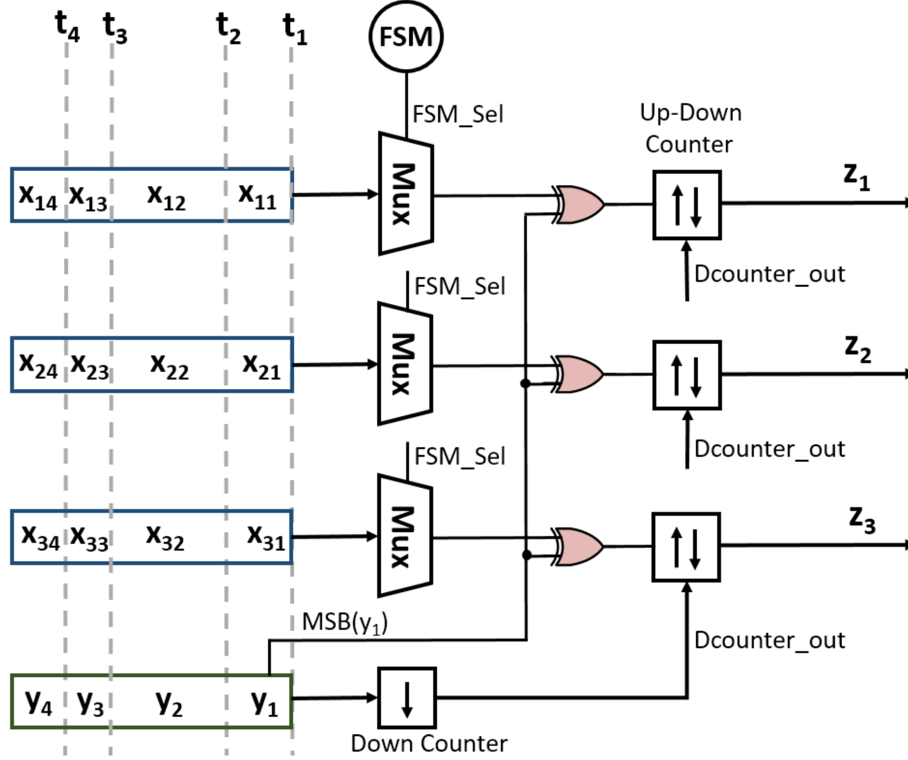


Figure 2.6: An Example of the MVM Module, in Which It Receives $X_{3\times4}$ and $Y_{4\times1}$ as Its Inputs and Generates the Output Vector $Z_{3\times1}$.

tion 2.5.2 to compute the components of the h and C vectors, respectively.

Sigmoid and Tanh Modules: The non-linear activation functions can be implemented in hardware using polynomial approximations Muller (2005), look-up tables, or CORDIC algorithms Hu *et al.* (1991). These implementations utilize large area and consume high power. Therefore, σ and $tanh$ in ELSA are implemented as piecewise linear functions Wang *et al.* (2017), as shown in Table 2.1, resulting in a more compact and lower power design.

Table 2.1: Piece-wise Linear Activation Functions Wang *et al.* (2017).

$\mathbf{HSig(x)} = \begin{cases} +1 & x > 2 \\ \frac{x}{4} + 0.5 & otherwise \\ 0 & x \leq -2 \end{cases}$	$\mathbf{HTanh(x)} = \begin{cases} +1 & x > 1 \\ x & otherwise \\ -1 & x \leq -1 \end{cases}$

2.5.7 Controller Units

Figure 2.7 shows the control flow graph (CFG) of the top-level controller (Top-C). It consists of three mini controllers – *MVM-C*, *EM-C* and *EMA-C*. The computation modules that involve variable-cycle operations (i.e. MVM, EM, EMA) require synchronization with other single-cycle operations (e.g. adders). Moreover, these units have to execute in parallel to maximize throughput. This cannot be accomplished by Top-C alone. The mini controllers are designed to individually control the AM-based units.

Top Controller (Top-C): This is responsible for synchronizing the AM-based modules with other single-cycle units and enabling parallel executions. As shown in Figure 2.7, it consists of 7 different states, where states $S1$, $S3$, $S5$ and $S7$ activate the MVM, EM and EMA modules. For example, when Top-C is in $S1$, the control token is passed to the MVM-C to start the MVM operations. The MVM-C operates on one set of data for multiple cycles and generates a complete detection signal that sends the control back to the Top-C. This is the case for all the Top-C states that call the mini controllers.

MVM mini-Controller (MVM-C): This activates the MVM modules and consists of two major states – *partial* and *full*. The *full* state is responsible for operating on all the columns of the matrix iteratively to compute the complete results. This state

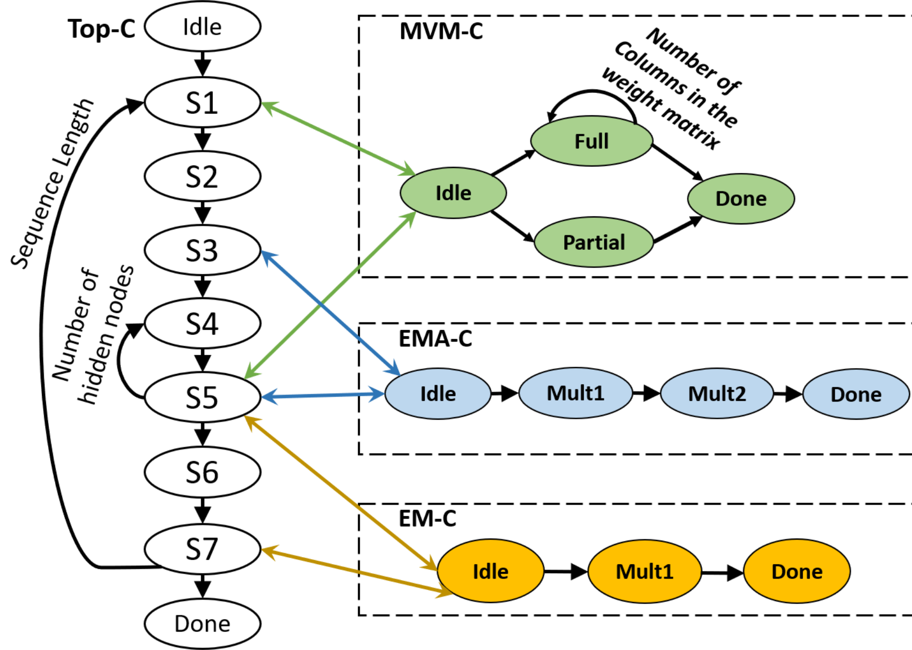


Figure 2.7: The Controller Hierarchy That Consists of a Top Controller (Top-C) and Three Mini Controllers- MVM-C, EMA-C and EM-C.

is used to generate the initial data for the pipelining flow. The *partial* state only operates on one column-scalar multiplication to generate one partial result. This state is designed to overlap its computation with the *EM* and *EMA* units which are active in $S5$ of Top-C.

EM mini-Controller (EM-C): The EM-C consists of one multiplication state to control the EM computation units. Once the operation is done, it sends the control back to Top-C, which then activates the MVM-C for overlapping the data computation in time steps $t + 1$ and t.

EMA mini-Controller (EMA-C): The EMA-C includes two consecutive multiplication states (i.e. Mult1 and Mult2) to activate the EMandAdder unit for generating one component of the memory state vector at each iteration.

2.6 Multi-level Elastic Pipelining

Some of the computation units in an LSTM network have data dependencies among themselves. These have to be executed sequentially, while others can execute in parallel. Although a non-pipelined version is straightforward, the throughput would be unacceptably low. Pipelining is essential and ELSA incorporates pipelining at *three* levels, involving variable-cycle multipliers, various computation units within the LSTM layer, and across multiple time-steps.

Table 2.2: This Shows the Control Flow and the Data Computation in the Proposed Pipelining Method. T Is the Total Number of Time Steps. n Is the Total Number of Hidden Nodes, and j Denotes the j_{th} Component of Its Corresponding Vector. The Output of the Stage Operations as Well as the Mode of Operation for the MVMs Are Specified Below. The Pipeline Stages Shown in Columns Are Executed in Parallel and the Stages Shown in Rows Are Performed Sequentially.

Multiple Cycles	**One Cycle**	**Multiple Cycles**	**One Cycle**	**Multiple Cycles**	**One Cycle**	**Multiple Cycles**
Stage 1 Ops (t) *full*	Stage 2 Ops (t) $\mathbf{f_j(t)}$, $\mathbf{\hat{C}_j(t)}$, $\mathbf{i_j(t)}$	Stage 3 Ops (t) $\mathbf{C_j(t)}$	Stage 2 Ops (t) $\mathbf{f_{j+1}(t)}$, $\mathbf{\hat{C}_{j+1}(t)}$, $\mathbf{i_{j+1}(t)}$ Stage 4 Ops (t) $\mathbf{o_j(t)}$ Stage 5 Ops (t) $\mathbf{tanhout_j(t)}$	Stage 6 Ops (t) → Stage 1 Ops (t+1) $\mathbf{h_j(t)}$, *partial* Stage 3 Ops (t) $\mathbf{C_{j+1}(t)}$	Stage 5 Ops (t) $\mathbf{tanhout_n(t)}$	Stage 6 Ops (t) $\mathbf{h_n(t)}$
Controller State 1	**Controller State 2**	**Controller State 3**	**Controller State 4**	**Controller State 5**	**Controller State 6**	**Controller State 7**

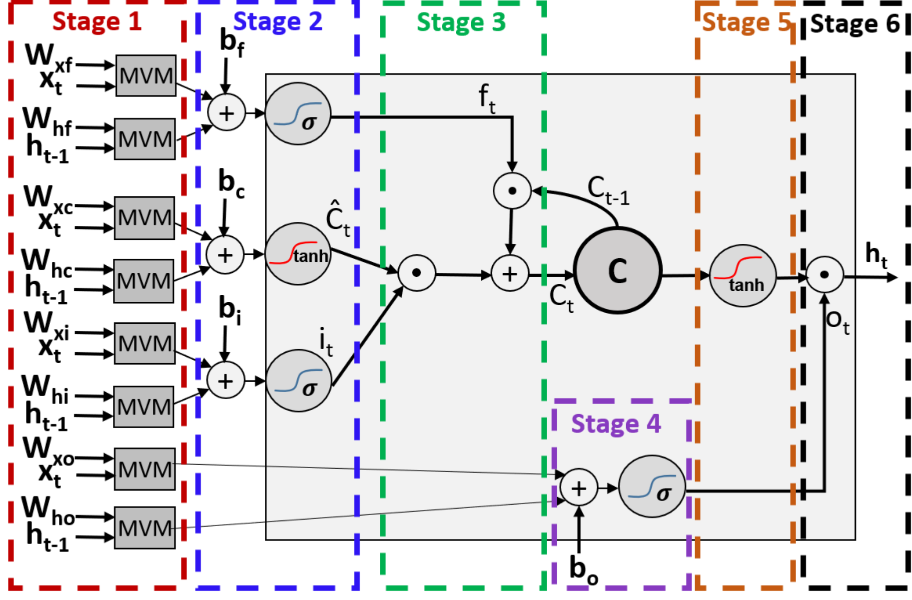


Figure 2.8: The Six Pipeline Stages in the LSTM Layer. *Stage* – 1: Eight Parallel MVMs; *Stage* – 2: Three Activation Functions and Ternary Adders; *Stage* – 3: Two Consecutive Multiplications and an Adder; *Stage* – 4: a Sigmoid Function and a Ternary Adder; *Stage* – 5: One Tanh Function; *Stage* – 6: One Element-wise Multiplication.

ELSA consists of six elastic pipeline stages as shown in Figure 2.8. The latency of some of these stages are multi-cycle and conventional pipelining methods are not efficient enough to maximize the throughput of this design. Table 2.2 shows the control flow of the pipelining method along with the data computations done in each controller state. The overlapping of the computation units starts in controller state 4 where the operations in pipeline stages 2, 4 and 5 at time step t are performed in parallel.

In controller state 5, the operations in stage-6 (time step t) and stage-1 (time step $t+1$) are overlapped with two consecutive multiplications in stage-3 (time step t). Since the stage-3 operations is independent of the ones in Stages 6 and 1, they can be executed in parallel. It is worth mentioning that with the proposed scheme,

the MVMs are almost completely overlapped with other units, as are the memory accesses, resulting in near maximum resource and memory utilization. All the intermediate results are written into the buffers so the SRAMs are only accessed for fetching the parameters and writing back the computed values for the hidden-state (h) and memory-State (C). These result in substantial reduction in the overall design latency as well as maximizing the throughput. These are quantified in Section 2.8 and Section 2.10.2.

2.7 Quantization and Accuracy

The objective of this section is to address the impact of employing the AM units on the final output of an LSTM network. While AM is accurate (see description in Section 2.5.1), it is still necessary to explore the impact of the bit-width and the feedback in LSTM on error propagation through the entire design. In addition, it is required to investigate whether the error accumulates in the hidden and memory states over various time-steps.

2.7.1 Model Description

This work is evaluated on a character-level language modeling (LM) network, which is one of the most widely used tasks in natural language processing Sundermeyer *et al.* (2015). This model predicts the next character given previous character sequences and generates a text character by character that captures the style and structure of the training dataset and generates a text that *looks like* the original training set. The LM used in this paper Karpathy (2016) is written in a scientific computing framework referred to as Torch. For the evaluation, the model is trained on a subset of Shakespeare's works by setting the batch size, training sequence and the learning rate to their default values of 50, 50 and 0.002, respectively Karpathy

(2016). The architecture of this network is shown in Figure 2.9.

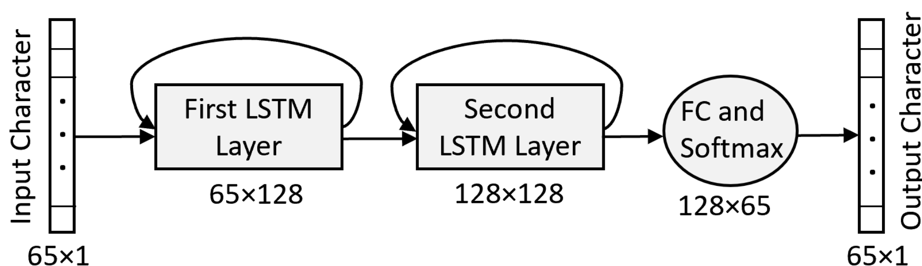


Figure 2.9: The LM Network. It Consists of Two 128 Hidden-node LSTM Layers Followed by FC and Softmax Layers. The Output of Both LSTM Layers Is a Vector of Size 128 and Their Inputs Are of Size 65 and 128, Respectively.

The number of characters used in this network is 65, which includes the lower and uppercase letters with some special characters. The input is a character formed in a one-hot vector of size 65. The first LSTM layer receives this input and outputs a hidden vector of size 128, which is fed as input to the second LSTM layer. Similarly, the second LSTM layer generates a 128-node hidden vector and passes the output to the fully connected (FC) and the softmax layer. The output of the final layer is a 65-node vector whose components represent the likelihood of that corresponding character being the output.

The primary operations in this model is matrix vector multiplications. The equations 2.1-2.6 describe the fundamental computations in the LSTM networks.

2.7.2 Quantized Model

Quantization is a method to improve the performance of the workloads for the inference phase. It involves reducing the number of bits representing a number. Typically neural network models use floating point 32 (FP32) to represent the weights and the activations. Model quantization converts FP32 numbers to a target lower precision integer data type and replaces the FP32 with the corresponding quantized

operation. In this work, quantization is applied to the trained LM model. This work is designed for the inference phase and training is performed off-line with FP32 in software using Torch7. Therefore, the accuracy measurements are for the inference phase. A range-based linear quantize operation is applied to the weights as described in Equation 2.9.

$$Q(W_{FP32}, n) = round(W_{FP32} - Min(W_{FP32}).scale) \tag{2.9}$$

$$scale = \frac{2^n - 1}{Max(W_{FP32}) - Min(W_{FP32})}. \tag{2.10}$$

W_{FP32} is the weight matrix in full precision and n is the number of target integer bits. The Q operation is applied to all the weight matrices and the bias vectors in the LSTM layers.

In this work, post-training quantization is performed on the model for 5 to 16 bits and the accuracy of the quantized model implemented with the AMs is then compared with both FP32 and Fixed-point exact implementations. For a fair comparison, the same input sequence x_t is fed to both FP32 and this work, where $t \in [1, 2, ..., 1000]$. The accuracy is computed as mean squared error (MSE) between this work and FP32. Due to the recurrent nature of the LSTM and the presence of the feedback on the memory state (C), and the hidden state (H), the MSE is calculated for both.

Figure 2.10 compares the accuracy of this work with FP32. Each bar is for a specific bit-width and the average MSE for H is computed over 1000 time-steps. In the 8-bit design, the magnitude of the MSE becomes stable and very close to zero. From 8 to 16-bit quantization, there is no significant change in the MSE. Thus, to achieve a good trade-off between the accuracy and the design metrics (i.e. power, area and performance), the weights and biases were quantized to 8 bits.

Figure2.11 illustrates the MSE (this work, FP32) of the hidden state for the 8 bit

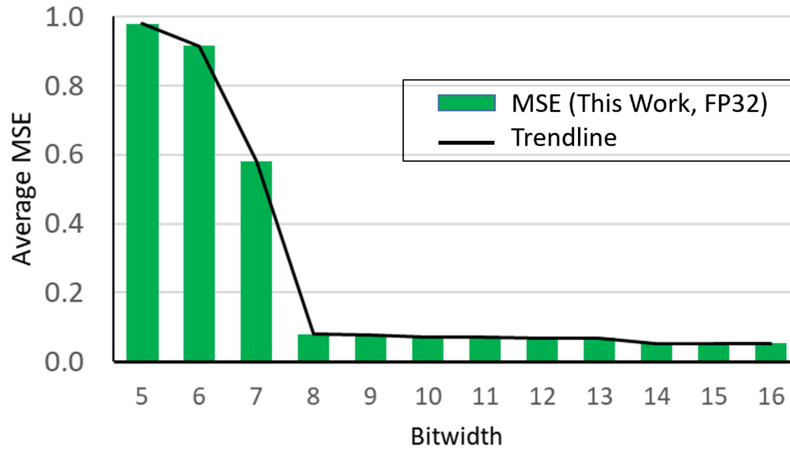


Figure 2.10: Average MSE for the Hidden State (H) Over 1000 Time Steps for Different Bit-width. The Memory State Demonstrated Similar Behavior, Hence Its Results Were Removed for Brevity.

quantization. The MSE is reported for 1000 time steps. The fluctuation in the MSE over the time-steps occurs because of the approximate nature of the AM and its data dependency, that rounds up/down the final product based on the inputs. This has the effect of canceling the errors. This shows that the error does not accumulate and does not grow exponentially when employing the AMs in the design.

To further investigate the accuracy of this work with the corresponding design with exact fixed-point multipliers, a thorough experiment is conducted to demonstrate how the accuracy changes from a single AM up to a network of LSTMs. Table 2.3 shows the accuracy for a single multiplication, a MAC unit, an LSTM layer and an application (i.e. LM that has two consecutive LSTM layers), when the AM is employed.

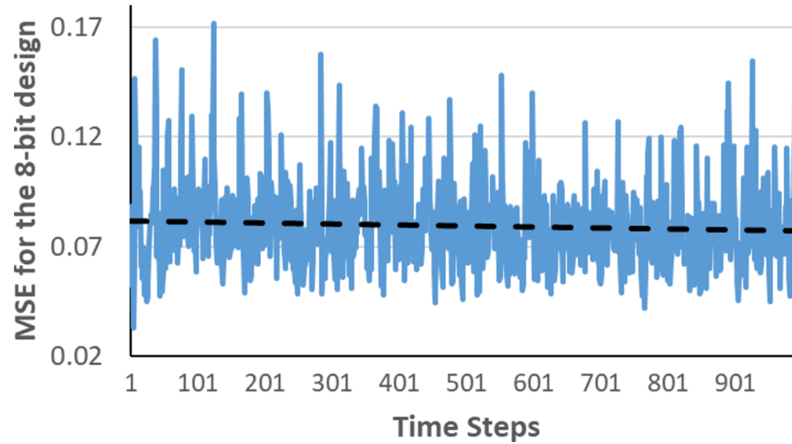


Figure 2.11: The MSE (This Work, FP32) of the Hidden State for the 8-bit Quantization. The x-Axis Demonstrates the MSE Values Over 1000 Time Steps. The Dashed Black Line Shows That the Error Does Not Accumulate. The Same Trend Is True for the Memory State for Which Its Results Are Removed for Brevity.

Table 2.3: The Relative Error for a Single Multiplication, MAC Operations and an LSTM Layer, as Well as the Classification Accuracy for an Application (i.e. LM That Has Two Consecutive LSTM Layers), When the AM Is Employed in the 8-bit Hardware Design.

	Relative Error
One Multiplication	1.5%
MAC Operations	2.2%
LSTM (one layer)	2.3%

	Classification Accuracy
Application (LM)	**96%**

The accuracy for one multiplication and MAC operation was computed as the fraction of differences between the 8-bit AM and its corresponding 8-bit fixed point exact multiplier. That is, for each input, the relative error of every pair of corresponding multipliers and MAC units was computed and these values were averaged over the set of applied inputs. The accuracy for one LSTM layer was measured as the average accuracy of the hidden states. The last entry of Table 2.3 reports the classification accuracy for LM. The accuracy degraded by 2.5% moving from an AM to a full application which consists of a network of consecutive LSTMs. The Top-5 classification accuracy (a standard measure particularly for LM) was 96%.

This work applies an 8-bit quantization on the weights and outputs of the model and stores the intermediate results up to 11 bits. The outputs of the MVM units are stored in 10 bits by increasing the bit-width of the up-down counters to avoid overflow. The outputs of the ternary adders are stored in 11 bits considering the one bit carry to preserve the precision. As the output of the activation functions is between -1 and 1, they can be stored in the initial 8-bit quantized format, thereby no truncation is required for the inputs of the EMs.

2.8 Performance Modeling for ELSA

This section presents a general model for the execution time of ELSA as a function of hidden nodes and time steps. This is to permit accurate evaluation of ELSA for any application that includes a network of LSTM layers, e.g., speech recognition, image captioning, etc. A similar performance model for the non-pipelined version of ELSA is also constructed to quantify the improvements due to the pipelining strategy employed in ELSA.

Let $X = (X_1, X_2, \cdots, X_T)$ and $H = (H_1, H_2, \cdots, H_T)$, where X_t and H_t are the input and output of ELSA at time step $t \in [1, 2, \cdots, T]$, respectively. In an LSTM

layer with N hidden nodes, $X_t = [x_t^1, x_t^2, ..., x_t^N]$ and $H_t = [h_t^1, h_t^2, ..., h_t^N]$.

As discussed in Section 2.6, each controller state may contain a single pipeline stage (e.g. controller state 2) or multiple pipeline stages (e.g. controller state 4). The execution time (D) of each controller state (CS) is denoted by D_{CS_i}, for $i \in [1, 2, ..., 7]$. The execution time is expressed in number of clock cycles. The operations performed in CS_2, CS_4 and CS_6 are single cycle operations whereas those in CS_1, CS_3, CS_5 and CS_7 are multi-cycle operations, whose latency is data-dependent and determined during run-time. The execution time of these operations is expressed in terms of the magnitude of their multiplicands (e.g. $\|x_t^j\|$ in stage 1 of Figure 2.8, where t denotes the time step and j is the j_{th} component of the X_t vector). This is because of the AM units, in which the multiplicands determine the execution time in number of clock cycles. In all the equations, $j \in [1, 2, ..., N]$, $t \in [2, ..., T]$ and i, o and f correspond to the input, output and forget gates, respectively. Note that the following equations can be directly derived from Table 2.2 and Figure 2.8.

2.8.1 Pipelined Design

The delay equations for ELSA with multi-level pipelining are shown in Equations 2.11-2.17, after the initial data is produced to flow through the pipeline stages (i.e t>2). The quantities in the equations correspond to the variables in Table 2.2. For example, since the MVM modules execute in parallel and X and H determine the execution time of these operations, D_{CS_1} in Equation 2.11 is the maximum value of each component of these vectors.

$$D_{CS_1}(t) = \max(\|x_{t+1}^N\|, \|h_t^N\|), \tag{2.11}$$

$$D_{CS_3}(t) = \frac{\|i_t^1\| + \|f_t^1\|}{2}, \tag{2.12}$$

$$D_{CS_5}(j,t) = \max\left(\frac{\|o_t^j\|}{2} + \max_{t\neq T}\left(\|x_{t+1}^j\|, \|h_t^j\|\right), \frac{\|i_t^{j+1}\| + \|f_t^{j+1}\|}{2}\right), \tag{2.13}$$

$$D_{CS_7}(t) = \frac{\|o_N^t\|}{2}, \tag{2.14}$$

$$D_{CS_2} = D_{CS_4} = D_{CS_6} = 1. \tag{2.15}$$

The total execution time of ELSA with pipelining ($D^p_{Total}(j,t)$), which is a function of hidden nodes and time steps is shown in Equation 2.16 and is simplified in Equation 2.17.

$$D^p_{Total}(j,t) = \sum_{t=2}^{T}(D_{CS_1}(t) + D_{CS_3}(t) + 2 + D_{CS_7}(t)) + \sum_{t=2}^{T}\sum_{j=1}^{N-1}(D_{CS_5}(j,t) + 1) \tag{2.16}$$

$$\begin{aligned} D^p_{Total}(j,t) = &\sum_{t=2}^{T}(D_{CS_1}(t) + D_{CS_3}(t) + D_{CS_7}(t)) \\ &+ \sum_{t=2}^{T}\sum_{j=1}^{N-1}(D_{CS_5}(j,t)) + T + N(T-1) - 1 \end{aligned} \tag{2.17}$$

2.8.2 Non-pipelined Design

The delay equations for the non-pipelined design are shown in Equations 2.18-2.22. Note that the same units and structure are used for both the designs. The only difference between these two designs are the way the operations are executed. In the non-pipelined version, the stages shown in Figure 2.8 execute in sequence. Hence, the execution time is expressed in terms of the pipeline stages, and does not correspond to the control sequence shown in Figure 2.7.

$$D_{stage_1}(j,t) = \sum_{j=1}^{N} \max(\|x_t^j\|, \|h_{t-1}^j\|), \tag{2.18}$$

$$D_{stage_3}(j,t) = \frac{\|i_t^j\| + \|f_t^j\|}{2}, \tag{2.19}$$

$$D_{stage_6}(j,t) = \frac{\|o_t^j\|}{2}, \tag{2.20}$$

$$D_{stage_2} = D_{stage_4} = D_{stage_5} = 1. \tag{2.21}$$

The total execution time of the non-pipelined design, which is denoted by $D^{np}_{Total}(j,t)$, is shown in Equation 2.22.

$$\begin{aligned} D^{np}_{Total}(j,t) = &\sum_{t=2}^{T}(D_{stage_1}(j,t)) + 3N(T-1) \\ &+ \sum_{t=2}^{T}\sum_{j=1}^{N}(D_{stage_3}(j,t) + D_{stage_6}(j,t)) \end{aligned} \tag{2.22}$$

To compute the impact of the pipelining method on the overall execution time of ELSA, equations 2.17 and 2.22 were evaluated and compared for different bit precision, hidden nodes and time steps. These are shown in Table 2.4. Thus, a total of 27 configurations were evaluated. Based on empirical data, the pipelining alone achieves 1.62X improvement in performance on average as compared to the non-pipelined design. The speedup achieved for each configuration was close to 1.62X, so only the average is reported.

Table 2.4: The Average Speed-up Achieved by the Pipelining Method over the Non-pipelined Design for *27* Different LSTM Configurations. This Was Computed by Evaluating Equations 2.17 and 2.22. The Minimum and Maximum Speedups Were 1.58x and 1.65x, Respectively.

27 Different LSTM Configurations				**Average Speedup (X) over 27 Configurations**
Bit Precision	8	12	16	**1.62X**
Hidden Nodes	64	128	256	
Time Steps	10	100	1000	

2.9 A Framework for Pre-hardware Mapping Analysis

This section describes a Python based framework that allows for analysis and fine tuning of the input parameters before mapping the design to hardware. Figure 2.12 depicts the overall structure of the framework. It consists of three main implementations: 1) **LSTM_AM:** an LSTM layer with the exact same architecture as proposed in Section 2.5 with the AM units; 2) **LSTM_FixedP_Exact:** an LSTM layer with the same design as LSTM_AM, except that the AM units are all replaced with exact fixed-point multipliers. Hence LSTM_FixedP_Exact does not include variable cycle operations and is exact; 3) **LSTM_FP32:** an LSTM layer that is the same as the original software implementation with FP32 operations. The input to this framework and the three implementations is a set of parameters, including the number of hidden nodes, weights and biases, number of time steps and bit-width. These implementations can be configured to implement any LSTM network topology. The main objective is to report the accuracy, speed up and the execution in number of clock cycles for the LSTM_AM design.

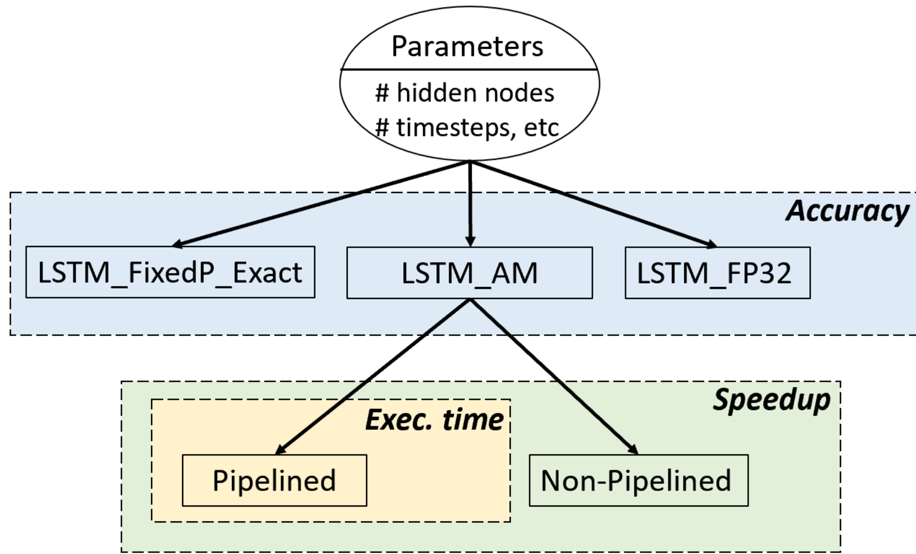


Figure 2.12: Overall Structure of the Framework. It Consists of Three Main LSTM Implementations in Python That Can Report the Accuracy of This Work for Different Bit-width, the Speed up Achieved with the Pipelining Method and the Execution Time of a Given Application in Clock Cycles.

2.9.1 Accuracy

To find the optimal bit-width for quantization, the accuracy of LSTM_AM needs to be compared against the other two implementations as described in the preceding section. The framework allows for a thorough exploration of the optimal bit-width for any given application with any network topology. A layer-wise analysis can also be done for improving the accuracy even further.

2.9.2 Speed up

The framework includes two versions of the LSTM_AM design: *pipelined* and *non-pipelined*. In the non-pipelined version, the stages shown in Figure 2.8 execute in sequence. This is to compute the impact of the pipelining method over the non-pipelined version and report the speed up for a given LSTM application with its time

steps, hidden nodes and parameters. This allows quantifying the improvements due to the pipelining strategy. For example, Table 2.4 presents the average speed up achieved by the 8-bit, pipelined design with two consecutive LSTM layers.

2.9.3 Execution Time

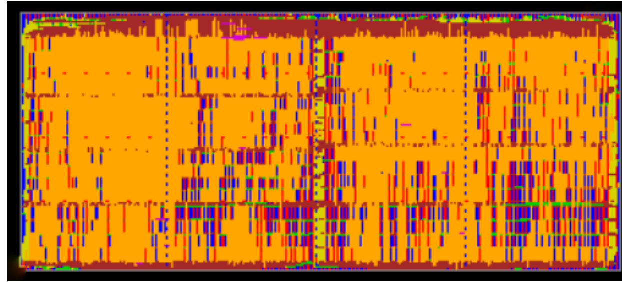
A general performance model of LSTM_AM with the pipelining strategy was also developed in this framework. This performance model is a function of hidden nodes and time steps with a predefined bit-width These are varied to evaluate the execution time of the application in terms of the number of clock cycles prior to mapping it into the hardware. This can be applied to any application that includes a network of LSTM layers, such as speech recognition and image captioning. This is invaluable as it can be an iterative procedure to fine-tune the input parameters to achieve a desirable execution time.

2.10 Experimental Results

2.10.1 ASIC Implementation of ELSA

ELSA's design was specified in RTL, synthesized and placed and routed (using Cadence® tools) in 65nm CMOS technology achieving a peak frequency of 322 MHz. ELSA's RTL design, including the controllers, is fully parametrized and can adapt to any LSTM network topology. Hence, there is no need to do the pipelining again as the controllers automatically accommodate the change. In addition, no design effort is required for varying the bit precision and modifying the size of the hidden nodes for a given application. Figure 2.13 shows the physical layout of ELSA's design in 65nm and the characteristics of the ASIC implementation. ELSA has sufficiently small cell area and low power, making it suitable for use in embedded systems. Moreover,

the efficient scheduling and pipelining techniques led to a design with high peak performance making ELSA also suitable for real-time applications.



	ELSA
Core Voltage (V)	1.1
Number of MACs	772
Precision (bit)	8-11
Frequency (MHz)	322
Total Cell Area (mm^2)	SRAM area: 2.22 LSTMCell area: 0.4
On-Chip Memory (KB)	106
Peak Performance (GOP/s)	27
Power (mW)	20.4
Energy-Efficiency (GOP/s/mW)	1.32

Figure 2.13: The Physical Layout of ELSA's Design in 65nm CMOS Technology (Top) and the ASIC's Implementation Results (bottom).

ELSA uses an 8-bit fixed-point representation (see Section 2.7) with the intermediate results extended to 11 bits to preserve the precision. The SRAMs incorporated in ELSA were provided by the 65nm library supplier. Unfortunately, the available SRAMs were larger than necessary and hence their area and power numbers shown

in Figure 2.13 should be considered as pessimistic, by as much as ∼6%. The SRAM area of ELSA is approximately 3X larger than its logic area. Clock gating of the computation units and the mini-controllers and the use of sleep modes for the SRAMs were employed to further reduce the power consumption. Because of the variable-cycle pipeline stages, ELSA's design greatly benefits from clock-gating. The greatest reduction in power was achieved when the computation units in a multi-cycle pipeline stage were maximally utilized. Hence, all the other idle units were clock-gated for several cycles.

Figure 2.14 shows the power and area breakdown of ELSA's components, including the SRAMs. The power consumption was measured using data activity information (*.vcd) obtained from the testbench by simulating the design after placement and routing. As expected, the SRAMs consume the most power. Among the submodules, the controllers contribute the least to the power consumption and the MVMs consume the most as there are 772 MACs in this design. There is substantial difference between the area utilization of the SRAMs and all the other components. Although there are 772 MACs in this design, the MVMs constitute to only 10.66% of the total area.

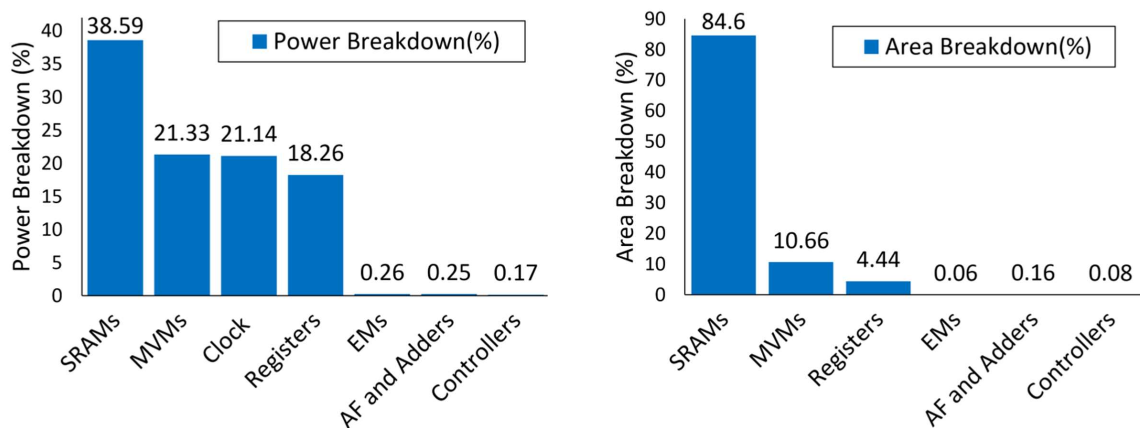


Figure 2.14: Power (Left) and Area (Right) Breakdown of ELSA's Components Including the SRAMs. AF Stands for Activation Functions.

Comparison with the Baseline-LSTM

The LSTM network was also designed with 8-bit exact fixed-point multipliers and is referred to as the Baseline-LSTM. This is functionally equivalent to ELSA except that all the AM units were replaced with the exact multipliers. These multipliers were optimally synthesized by the Cadence tools (i.e. Genus) based on the clock frequency constraint. This is automatically generated by Genus to meet the timing constraints corresponding to the given clock frequency. The Baseline-LSTM was also specified in RTL and synthesized and placed and routed in 65nm technology. The ASIC implementation results of ELSA are compared with the Baseline-LSTM and the normalized results are shown in Figures 2.15 and 2.16. In Figure 2.15, ELSA and the Baseline-LSTM were run at the same clock frequency (the peak frequency of the Baseline-LSTM). The energy efficiency ($GOP/s/mW$) and area efficiency ($GOP/s/mm^2$) of ELSA exceeds that of the Baseline-LSTM by 1.2X. The cell area and power consumption of ELSA are also lower (0.3X), but the peak performance of the Baseline-LSTM is higher by 3.3X. This is to be expected as the operations in the Baseline-LSTM are single cycle operations and the Baseline-LSTM was run at its highest clock frequency, unlike ELSA's.

For a thorough comparison, both designs were also run at their individual maximum achievable clock frequencies. The results are shown in Figure 2.16. Due to the compactness of the compute-intensive units of ELSA, which are on the critical path, ELSA can run 3.2X faster in terms of clock frequency. While the ratio of the energy efficiency is maintained at 1.2X moving from Figure 2.15 to Figure 2.16, the area efficiency of ELSA is greatly improved and reaches 3.6X. This is mainly due to the increase in the peak performance as the increase in the cell area was negligible and the ratio remains at 0.3X. Although running ELSA at its highest clock

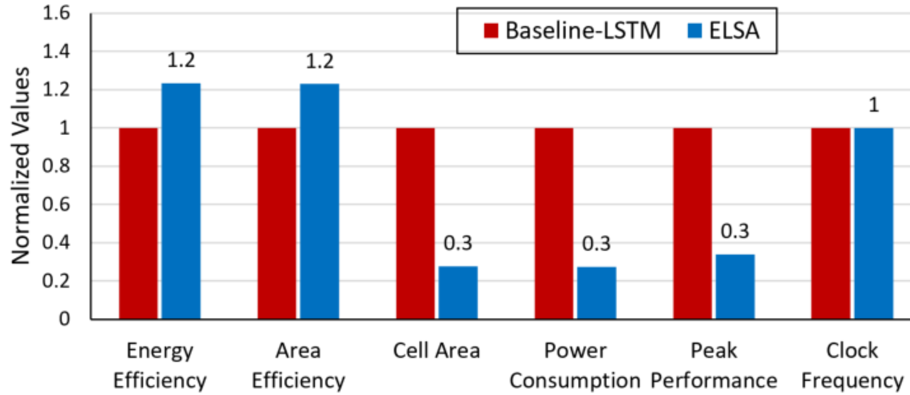


Figure 2.15: The ASIC Implementation Results of ELSA as Compared to the Baseline-LSTM. Both of These Designs Were Run at the Same Clock Frequency, the Highest That the Baseline-LSTM Can Achieve. The Reported Numbers Are Normalized. The Energy and Area Efficiency of ELSA Exceeds That of the Baseline-LSTM by Factors of 1.2x.

frequency increased its power consumption, it is still lower (0.9X) than that of the Baseline-LSTM.

Comparison with the Existing ASIC Implementations

ELSA is also compared against the existing ASIC implementations of LSTMs – DNPU Shin *et al.* (2017) and CHIPMUNK Conti *et al.* (2018) as shown in Table 3.2. DNPU is a CNN-RNN processor and its application requires a combination of CNNs and RNNs. CNN is its major component and RNN was not evaluated as a standalone component. Although ELSA has twice the bit-precision and uses 10X more SRAMs than DNPU, it achieves higher peak-performance and consumes less power. DNPU's bit-width (4-bits) is half of ELSA's. Scaling ELSA to 4-bits would increase the peak-performance (at-least 54 GOPs) and the frequency (∼400MHz), and decrease the power consumption. These would lead to a much higher energy-efficient design. In addition, DNPU has only 10KB of on-chip memory which limits its peak-performance

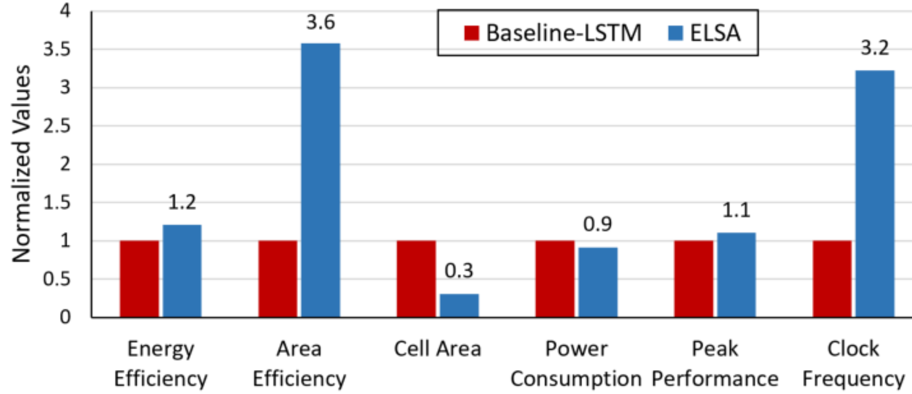


Figure 2.16: The ASIC Implementation Results of ELSA as Compared to the Baseline-LSTM. Both of These Designs Were Run at Their Highest Achievable Clock Frequency. The Reported Numbers Are Normalized. The Energy and Area Efficiency of ELSA Exceeds That of the Baseline-LSTM by Factors of 1.2x and 3.6x, Respectively

by requiring the use of external memory even for small networks. The application in which the functionality of ELSA was evaluated on (even for 4-bits), does not fit on DNPU and requires a DRAM. This lowers DNPU's peak-performance substantially. CHIPMUNK uses 22% smaller SRAMs. It achieves higher peak-performance, but it consumes 30% more power, making ELSA more energy-efficient. As shown in the last entry of Table 3.2, ELSA's energy-efficiency exceeds that of DNPU and CHIPMUNK by 1.2X and 1.18X, respectively.

Table 2.5: Comparison with the Previous ASIC Implementations. All of These Implementations Are in 65nm Technology. DNPU Is a CNN-RNN Processor, and This Table Only Includes the RNN Values Reported in Shin *et al.* (2017). The LSTM Architecture of DNPU and CHIPMUNK Differ Substantially among Themselves and Also When Compared with ELSA. Moreover, the Reported Applications Are Dramatically Different, Making Comparisons in General Difficult to Judge.

	DNPU Shin *et al.* (2017)	**CHIPMUNK Conti *et al.* (2018)**	**ELSA**
Precision (bit)	4-7	8-16	8-11
Frequency (MHz)	200	168	322
On-Chip Memory (KB)	10	82	106
Peak Performance (GOP/s)	25	32.3	27
Power (mW)	21	29.03	20.4
Energy-Efficiency (GOP/s/mW)	1.10	1.11	**1.32**
ELSA's Energy-Efficiency (X)	**1.2**	**1.18**	1

2.10.2 FPGA Implementation of ELSA

This work was implemented on Xilinx Zynq XC7Z030 FPGA which consists of 125K logic elements, 78.6k LUTs, 400 DSP blocks, 265 BRAMs and 157.2k FFs. This RTL design was synthesized and then implemented by Vivado Design Suite 2017.2. Table 3.2 provides a detailed comparison of several design metrics of this work with prior FPGA implementations for LM. It should be noted that all the three existing LM implementation use the same network as shown in Figure 2.9. After the computations in the first and second LSTM layers are completed, the outputs of the model are read from the BRAMs and are verified by comparing them against the validation data.

The power consumption of this work on the Zynq XC7Z030 FPGA is estimated by the Vivado Power Analysis tool using the post-placed netlist. This work does not use any DSP blocks in the FPGA since it takes advantage of the design of the AM to increase the energy-efficiency. Not using available resources might be viewed as inefficient. On the contrary, this is an advantage, as it makes it possible to implement the proposed design in low-cost, ultra low-power non-DSP commercial FPGAs such as Lattice Semiconductor (2021) (see section 2.10.3 for more details).

Table 2.6: Comparison with Previous Language Modeling Implementations.

	Chang *et al.* (2015)	**Chang and Culurciello (2017)-DeepStore**	**Chang and Culurciello (2017)-DeepRNN**	**This Work**
Application	Language Modeling	Language Modeling	Language Modeling	Language Modeling
FPGA	Xilinx Zynq XC7Z020	Xilinx Zynq XC7Z020	Xilinx Zynq XC7Z020	Xilinx Zynq XC7Z030
Technology (nm)	28	28	28	28
Design Entry	C-language	C-language	C-language	RTL
Frequency (MHz)	142	142	142	100
Precision (bits)	16 fixed	16 fixed	16 fixed	8 fixed
DSP Utilization	50 (23%)	NA‡	NA	0
LUT Utilization (K)	7.6 (11%)	NA	NA	23 (30%)
BRAM Utilization (36KB)	16 (12%)	NA	NA	180 (68%)
Power (W)	1.94	2.3	1.8	1.19
Throughput (GOPS)*	0.29	1.05	0.73	8.08/2.26
Energy Efficiency (GOPS/W)†	0.15	0.46	0.40	6.79/1.89

* The throughput of this work is data-dependent. 8.08 is the maximum throughput that it achieves and 2.26 is the average throughput.

† The energy-efficiency of this work for the maximum throughput is 6.79 and for the average throughput is 1.89.

‡ NA = Not Available

Table 2.7 shows the maximum **improvement** of this work over the existing implementations in throughput and energy-efficiency. The overall throughput of this work outperforms the state-of-the-art designs, Chang *et al.* (2015), DeepStore Chang and Culurciello (2017) and DeepRNN Chang and Culurciello (2017) by 27.86X, 7.69X and 11.06X, respectively.

Table 2.7: Maximum **Improvement** in Throughput and Energy Efficiency as Compared to Prior Implementations.

	Chang *et al.* (2015)	**Chang and Culurciello (2017)-DeepStore**	**Chang and Culurciello (2017)-DeepRNN**
Throughput (X)	27.86	7.69	11.06
Energy Efficiency (X)	45.26	14.76	16.97

This work is also more energy-efficient than the existing implementations by factors of 45.26X, 14.76X and 16.97X, respectively. Table 2.8 shows the resource utilization.

Table 2.8: Resource Utilization of Our Accelerator.

	Utilization	**Available**	**Utilization(%)**
LUT	23,036	78,600	29.31
FF	28,481	157,200	18.12
BRAM (36KB)	180	265	67.92

Table 2.9 compares the energy-efficiency of this work with CPU and GPU implementations. The Torch7 implementation of language modeling Karpathy (2016) is executed on the CPU and GPU platforms. The benchmark platforms in Chang and Culurciello (2017) are not the state-of-the-art. However, even a $10X$ improvement

over these platforms show that the CPU and GPU are substantially less energy-efficient than this work.

Table 2.9: Energy Efficiency of Different Platforms.

Platform	**CPU Chang and Culurciello (2017)**	**GPU Chang and Culurciello (2017)**	**This Work**
Energy-Efficiency (GOPS/W)	0.009	0.02	6.79/1.89

2.10.3 Summary of the Key Features of This Work

Scalability: The RTL design of this work including its controllers is fully parametrized and can adapt to any LSTM network topology. There is no need to do the pipelining again as the controllers automatically accommodate the change. Therefore, no design effort is required for varying the bit-width of the operands and modifying the size of the network.

Memory-Access Time: As this work is highly pipelined at multiple levels, most of the memory access time is overlapped by other computation units and the resources are kept at being maximally utilized. Moreover, all the intermediate results are written into the buffers so the BRAMs are only accessed for fetching the parameters and writing back the computed values for the hidden-state (H) and memory-State (C).

Advantages of Not Using DSPs: There are two types of non-DSP FPGAs that benefit from this work. The first one is to map this work directly onto one of the ultra low-power non-DSP lattice FPGAs Lattice Semiconductor (2021). This results in even lower cost and power implementation. An open-source FPGA Liu (2014) has recently been developed, which is parameterizable and user expandable with academic tool-flow support- VPR Betz and Rose (1997) and VTR Rose *et al.* (2012). Hence,

the second way is to exploit this and customize our FPGA all the way to the layout without adding extra level of complexity caused by DSPs. This enables OpenFPGA design and reduces the design complexity. The low-cost AM in this work can also be designed as a hard IP using OpenFPGA. It should be noted that both of these low-cost FPGAs are very well suited for being incorporated into SoCs.

2.10.4 Opportunities for Further Improvements

The main objective of this work was to design a compact, low power LSTM processor. One important additional mechanism that is part of our future work is *weight compression*. This would result in further significant improvement in area and power. There has been an extensive body of research on reducing the memory footprint for various types of neural networks Han *et al.* (2015); Xu *et al.* (2018). Chief among them are techniques that compress weights. In fact, the size of the required memory can be reduced by up to 90% Han *et al.* (2017). These techniques can also be applied to this work, in which case, the relative improvement of this work would be substantially higher.

2.11 Chapter Summary

This paper presents a novel scalable LSTM hardware accelerator, referred to as ELSA, that results in small area and high energy-efficiency. This is due to several architectural features, including the use of an improved low-power, compact approximate multiplier in the compute-intensive units of ELSA, and the design of two levels of controllers that are required for handling the variable-cycle multiplications. Moreover, ELSA includes efficient synchronization of the elastic pipeline stages to maximize the utilization. ELSA achieves promising results in power, area and energy-efficiency making it suitable for use in embedded systems and real-time applications. This ac-

celerator can be further improved by incorporating more compact SRAMs to achieve a more optimized floor-plan. In addition, the energy-efficiency can be significantly improved by applying weight compression techniques.

Chapter 3

AN ALTERNATE DATA-DRIVEN SURROGATE MODEL FOR UNCERTAINTY QUANTIFICATION

3.1 Problem Background: The Need for Uncertainty Quantification and Sensitivity Analysis

Consider a data-driven surrogate model (see section 1.2) defined as, $x \in \mathcal{D}_x \mapsto y = \widetilde{\mathcal{M}}(x;\theta)$, where x is the input and θ is a set of parameters. In many real-world problems, the inputs are random quantities and their value depend on outcomes of a random phenomena, i.e. they belong to a stochastic domain. Thus, the inputs must be viewed as distributions rather than singular quantities. For example, an autonomous vehicle frequently has access to only sparse and uncertain parameter estimates which are drawn from sensors such as LIDAR and vision. This is referred to as input uncertainty. In this example, the vehicle parameters such as fuel/electricity consumption and mechanical wear could also be uncertain. Hence, various sources of uncertainty exist that could arise due to the errors in the input data measurements, the model chosen to approximate the input to output mapping, the choice of the parameter values, etc. An end-to-end analysis of errors due to the imperfect models with uncertain input and model parameters is referred to as uncertainty quantification (UQ).

Let $f_X(x)$ denote the p.d.f of a random variable x. It is used to model the randomness in the inputs x. The objective of uncertainty propagation (UP) from the input is to estimate the statistics of the output $Y = \mathcal{M}(X)$, where X and Y are the input and output random variables, respectively. Figure 3.1, adapted from Sudret

(2007); Sudret and Kiureghian (2000), depicts different types of UP methods. Response variability methods estimate the dispersion of the system response around a mean when the inputs also vary around their mean. These methods are a type of UP and the first two moments of the output response are provided and evaluated (i.e. mean and the variance). *Perturbation method*, *weighted integral method* and *quadrature method* are examples of this type Sudret and Kiureghian (2000). The objective of structural reliability methods is to determine the tail probability, which represent the probability that a structure does not perform satisfactorily within a given period of time and stated conditions Sudret (2007); ChangWu Huang (2017). This is performed by computing the probability of exceeding a predetermined threshold and is referred to as the probability of failure. The third class of methods is referred to as spectral methods Ghanem and Spanos (1991) that provide the means to efficiently compute the empirical joint p.d.f of all the outputs. With these methods, the problems related to second moment and structural reliability methods can be solved by a straightforward post-processing of their output Sudret (2007); Sudret and Kiureghian (2000).

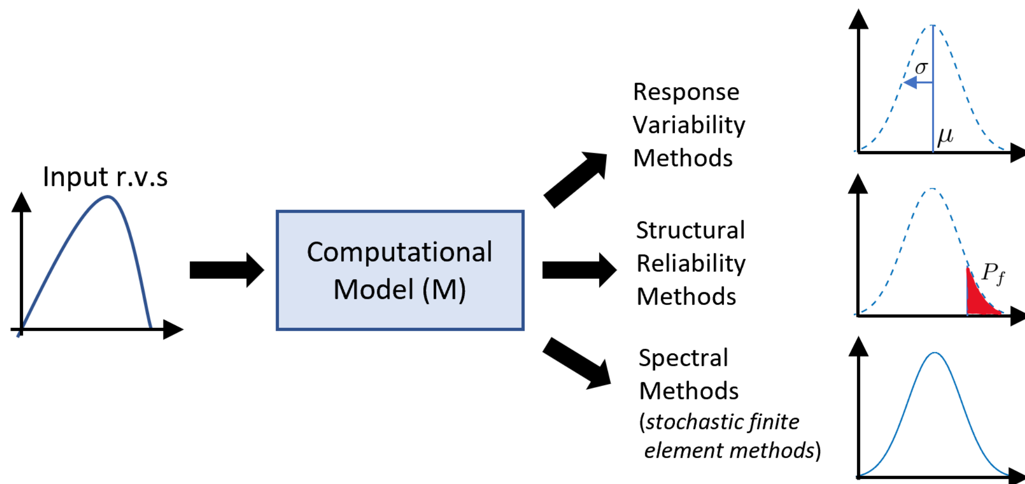


Figure 3.1: Different Types of Uncertainty Propagation Methods.

Modeling complex systems usually involves a large number of input parameters. In many problems, it may be the case that only a subset of the inputs influence the output response Y significantly. Global sensitivity analysis (GSA) refers to quantifying the sensitivity of the output to some or all subsets of the inputs. The objective of the GSA is to select the important input parameters and is usually performed by defining an importance measure. GSA would lead to reducing the model complexity by eliminating subsets of input variables that are not essential to the output.

Deep learning have achieved excellent results in accuracy and performance for perception, cognition and problem solving. However, in many real-world applications, uncertainty plays a significant role in the networks output prediction. Commonly, the approximation of Bayesian methods is used for estimating the uncertainty of the prediction, however, these can still have difficulties to reliably approximate the uncertainty of their output. Moreover, deep neural networks cannot provide the sensitivity of the predicted output with respect to its inputs. This results in having redundant nodes in the networks that increase the computational cost and response time, and hence reduces the flexibility to deploy these networks for real-time and safety-critical applications.

This work develops a robust framework for constructing data driven models using supervised learning. The surrogate model that this work employs in the presence of input variability is referred to as *arbitrary polynomial chaos expansion (aPC)* which is a data-driven model based on an extension to *polynomial chaos expansion (PCE)*. This is a new approach to training and inference in machine learning (ML). This approach estimates or learns a function that best explains the given set of input-output pairs. This is viewed as selecting one function from among an infinite, uncountable space of continuous functions. However, unlike all existing approaches, the proposed approach does not impose a probabilistic model on the function space. Instead, it

represents the elements of the function space as multivariate orthogonal polynomials in the underlying input variables. In such a representation, known as *aPC*, the polynomials are constructed directly from the moments of the inputs. These polynomials serve as the basis functions for representing the elements of the function space. In contrast to many existing approaches, no assumptions (e.g., Gaussian processes or prior distributions on the parameters, etc) are made on the elements of the function space. The proposed method also provides a direct method to compute the sensitivity of the model output with respect to any subset of input variables, providing a systematic and incremental method to pruning the model or changing the order of the model. The key advantages of this method are summarized below.

1. In addition to point-wise prediction, this approach provides an estimate of the first two moments of the output. Particularly, uncertainty in this method is a simple function of the expansion coefficients of the model, which depend on the raw data. As in other methods, the mean here is also used as the point estimate, and sigma is considered as an estimation to statistical uncertainty.

2. Higher order moments of the output and its p.d.f can be estimated using Monte Carlo (MC) methods efficiently. This can be done by evaluating the constructed aPC model for sufficient samples of input. In this approach, computation of the p.d.f is very efficient as evaluating the aPC method is not compute-intensive.

3. GSA can also be used to perform inference in using low accuracy models first, and then switching to more accurate models as the need arises. This approach provides a direct method to reducing the complexity of the model or changing the order of expansion. Particularly, it can rank the model inputs according to their influence on the variance of the output. This leads to eliminating the least important inputs and reducing the model complexity.

4. There is no need to tune hyper-parameters as in NN models and the model can be constructed with limited training data. These are some of the reasons that lead to substantial reduction in computations compared to existing techniques.

This framework is proposed as a general model that can be applied to any regression and classification workloads. The main contributions of this work are summarized below.

1. A data-driven methodology based on an extension to PCE is re-established as a framework for regression tasks, also performed by NN models. In addition to the point-wise prediction, this framework can provide a measure for quantifying the uncertainty in the output.

2. This methodology is then extended to address the scalability issue in tasks with large number of input variables. Moreover, it is extended for discrete outputs so that it can be applied on classification tasks similar to NN models.

3. This method is also turned into an executable computation graph. The structure of this graph intuitively demonstrates the sequential and parallel operations. This leads to even faster training and inference time. All the nodes at one level can be executed simultaneously, while the tasks from top to bottom are data dependent and are run in sequence.

4. The accuracy of this framework can be improved by constructing a higher order model. It is shown that this can be done by incrementally adding to the computation instead of recalculating all the polynomial basis and the coefficients. This allows a much faster training.

We demonstrate that our framework achieves point-wise predictions comparable to that of the existing NN models. The comparisons are presented on benchmark

datasets widely used in the literature. In addition, this framework provides: 1) a direct measure for quantifying the uncertainty in the model output by estimating its first two moments. The p.d.f of the output can also be estimated by using MC. 2) the sensitivity of the output can be estimated with respect to any subsets of the inputs. This helps with pruning the model and reducing its complexity.

Section 1.3 described NN models as well-known data-driven surrogate models. The following sections explain the challenges in UQ when employing NN models and then present an alternative data-driven surrogate model referred to as *polynomial chaos expansion* to address the challenges.

3.2 Challenges in UQ with NN Models

Although deep learning models are increasingly being employed in autonomous vehicles, medical diagnosis, robotics, and other safety critical applications, the reported successes hide a severe threat–the lack of an accurate measure of uncertainty associated with the prediction. Hence, UQ has regained attention particularly in the deep learning domain and new methods have been proposed. In prediction, UQ allows for the handing over control to a human or transitioning to a safe mode by an autonomous vehicle/robot or suggesting further analysis in medical diagnosis. The followings briefly discuss the related work.

Classical Bayesian Techniques: Bayesian inference (BI) Ghahramani (2015); Jaynes (2003) is a statistical inference method which is used for prediction and UQ. Extensive research has been done in the field of Bayesian machine learning as well Ghahramani (2015).

Figure 3.2 illustrates an example of BI, in which the posterior distribution is generated by multiplying the prior and likelihood distributions. In BI, a prior probability distribution, $P(\theta)$, is assumed as the initial uncertainty in the parameters θ, before

processing or realizing the input data. Then, the likelihood function, $P(data|\theta)$, is constructed which is the conditional probability of the observed data, given θ values. The final step is to the construct the posterior distribution, $P(\theta|data)$, by multiplying the prior and likelihood distributions, derived from Bayes' theorem. The posterior is the conditional probability of θ, given the observed data. The mean of the posterior is typically considered as the point estimates and the spread of the distribution is considered as an estimation to statistical uncertainty.

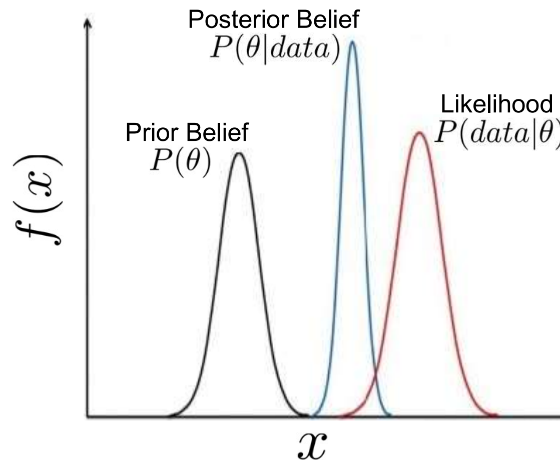


Figure 3.2: An Example of Bayesian Inference, in Which the Posterior Distribution Is Generated by Multiplying the Prior and Likelihood Distributions.

The key challenge in BI is the computation of the posterior. Sampling-based methods are the most widely used for computing the posterior. One class of such techniques is referred to as Markov chain Monte Carlo (MCMC) Metropolis *et al.* (1953); HASTINGS (1970). This is used to approximate the posterior distribution of the parameter of interest, θ, by systematic random sampling from a high-dimensional probability distribution. The key idea is to construct a Markov chain and draw samples such that the next sample is dependent on the current one, unlike Monte Carlo. Due to the auto-correlation of the samples, this class of methods is less efficient

as compared to Monte Carlo simulation. Moreover, employing these methods leads to a high computational cost that is often infeasible, particularly for deep neural networks with millions of parameters.

Bayesian Neural Networks: In general, neural networks are powerful surrogate models that can provide excellent results on point-wise predictions. To enable UQ, BI calculates the posterior distribution of the network parameters, $P(\theta|data)$, given the training data. This is intractable for neural networks, particularly deep models. Progress in variational inference (VI) has revived Bayesian neural networks and been widely used Kingma and Welling (2014); Blundell *et al.* (2015). VI provides an analytical approximation to $P(\theta|data)$ and are faster than classical sampling methods, such as MCMC Hinton and van Camp (1993); Graves (2011). VI finds the parameters of the distributions on the weights that are the closest to the actual posterior of the weights. Kullback-Leiber (KL) divergence determines the closeness.

Gal and Ghahramani (2016) suggested that using dropout during inference in deep learning can be interpreted as Bayesian approximation of the Gaussian process Rasmussen and Williams (2005), a well-known probabilistic model. Dropout is a method used in the training phase in which individual neurons are either eliminated (along with their incoming and outgoing edges) with probability of $1-p$ or kept with probability of p during forward and backward passes. This method is shown to be a useful regularization approach to avoid overfitting (i.e. closely fit to the training data) Srivastava *et al.* (2014). In Gal and Ghahramani (2016), it was shown that dropout integrates over the model parameters approximately, and no change in the model architecture and optimization is needed. Particularly, for a given input, to approximate the posterior distribution, the inference phase is executed several times while sampling from the weight space in every iteration. Figure 3.3 depicts one example of the weight sampling method for diabetic retinopathy detection, which is a

binary classification tasks with two outputs, healthy or diseased Leibig *et al.* (2017). The deep NN models in this example is a network similar to VGG-16 Simonyan and Zisserman (2014), in which dropout is added after each convolutional layer with probability of 0.2. The predictions are denoted by μ_{pred} and the uncertainty is quantified by σ_{pred}. The left most example shows the highest confidence while the right most example reports the highest uncertainty.

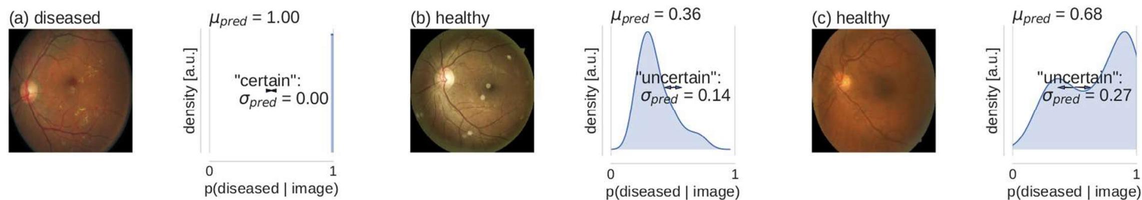


Figure 3.3: Implementation of Weight Sampling Method in Deep Learning Models for Diabetic Retinopathy Detection Leibig *et al.* (2017). The Images Are Examples of Fundus Images with Human Label Assignments on Top. Corresponding to Each Image Is the Approximate Predictive Posteriors over the Softmax Output Values $P(diseased|image)$. Predictions Are Based on the Mean of the Posteriors (i.e μ) and Uncertainty Is Quantified By σ.

One of the drawbacks of MC dropout is that it is mainly limited to convolutional layers and architectural choices need to be made Mukhoti and Gal (2018). Moreover, MC dropout requires thousands of iterations for getting a smooth posterior and is infeasible for real-time scenarios. In addition to the compute-intensive nature of approximating the posterior and hence UQ, deep learning models have another significant drawback, that is the difficulty in analyzing the global sensitivity of the output to any subset of the inputs. This often results in constructing large models with redundant nodes that increase the computational cost and response time, and reducing the flexibility to deploy these networks for real-time and safety-critical applications. This is the main reason that techniques such as compression and quantization are

often employed to reduce the size of the models (see section 1.5.1).

Ensemble Methods: Ensemble methods have achieved excellent results in quantifying uncertainty and are widely used in various domains Beluch *et al.* (2018). Lakshminarayan et al. Lakshminarayanan *et al.* (2017) propose an alternative to BNNs to estimate the predictive uncertainty. They proposed to train an ensemble of networks each with random initialization of the network parameters. The networks were all trained random shuffling of the entire training dataset to obtain good performance.

The main drawback of this approach Lakshminarayanan *et al.* (2017) is the high computational cost that comes with training multiple networks. This cost becomes even more significant in deep network architectures used for object detection, semantic segmentation, etc. Similar to MC dropout, this method does not provide any insight on the sensitivity of the output with respect to every input and the interactions between them, which leads to redundancy in the networks. The following sections describe an alternate data-driven surrogate model for addressing UQ. This is a new approach to training and inference that provides a method for GSA as well.

The next section describes an alternate data-driven surrogate model for addressing UQ. This is a new approach to training and inference that provides a method for GSA as well.

3.3 Overview of Polynomial Chaos Expansion

When analyzing a physical system, a key task is to understand the relationship between the inputs and outputs. When there is stochasticity in the inputs, the system's response will also become a random quantity. Typically, a system with a set of inputs, X, and outputs, Y, is given in some form of an implicit relation $F(X, Y, \xi)$, where ξ is a set of parameters that are subject to variations. X, Y and ξ can be vectors of arbitrary dimensions.

Polynomial chaos expansion (PCE) Ghanem and Spanos (1991); Soize and Ghanem (2004); Augustin *et al.* (2008) is one approach to approximate Y by replacing the implicit relation $F(X, Y, \xi)$ with an explicit function $\widetilde{Y}(X, \xi)$. PCE is a representation of a 2nd order stochastic process as a multivariate orthogonal polynomial over an infinite dimensional Hilbert space. The first two moments of the expansion converge to that of Y in limit. The classical PCE was originated from the Wiener's work Wiener (1938) and is known as *Wiener-Hermite expansion*. It represents a random variable with an infinite series of Hermite polynomials in independent Gaussian random variables as shown in equation A.1.

$$Y(\xi) = \sum_{i=1}^{\infty} c_i H_i(\xi), \tag{3.1}$$

where c_i are the coefficients of the Hermite polynomial basis functions, $H_i(\xi)$. In practice, this sum is truncated to a limited number of basis functions, as expressed in A.2.

$$\widetilde{Y}(\xi) = \sum_{i=1}^{M} c_i \Phi_i(\xi), \tag{3.2}$$

where $\Phi_i(\xi)$ are the basis functions (determined based on a selected method) and c_i are the coefficients that need to be solved. The coefficients c_i can be obtained by Galerkin projection method. This method determines the function $\widetilde{Y}$ in such a way that the error $(Y - \widetilde{Y})$ is orthogonal to the space where Y belongs to.

PCE is not restricted to Gaussian processes and has been generalized for other standard distributions. Xiu and Karniadakis Xiu and Karniadakis (2002) introduced generalized polynomial chaos (gPC) involving non-Gaussian random parameters. The optimal basis is dependent on the underlying distribution of $\boldsymbol{\xi}$, i.e. the Hermite

polynomials are replaced by the sequence of optimal orthogonal basis functions with respect to the probability distribution of $\boldsymbol{\xi}$.

3.4 A Data-driven Framework

Previous section explored the model construction for the case in which the distribution of $\boldsymbol{\xi}$ are assumed or known. However, the underlying distributions of the parameters in practice may not be known or cannot be determined. This issue was resolved by the method proposed by Oladyshkin et al. Oladyshkin and Nowak (2012) and is referred to as *arbitrary polynomial chaos* (aPC). aPC is a data-driven method which extends gPC by estimating the probability measure using a finite number of moments.

Let $\boldsymbol{\xi} = \{\xi_1, \xi_2, ..., \xi_N\}$ be a set of N input random variables. This section describes the aPC method which is employed to construct the function $Y(\boldsymbol{\xi})$. The output $Y(\boldsymbol{\xi})$ is expressed as a weighted linear combination of the basis polynomials as in equation A.13.

$$Y(\boldsymbol{\xi}) = \sum_{i=1}^{M} c_i \Phi_i(\boldsymbol{\xi}), \tag{3.3}$$

where $\Phi_i(\boldsymbol{\xi})$s and c_is are the orthonormal multivariate basis polynomials, and the coefficients. The total number of coefficients and the multi-variate polynomials are $M = \binom{N+d}{d}$, where d is the degree of the expansion. The coefficients c are computed using mean-squared (cost function $J(c)$). This is expressed in equation 3.4, in which $Y_s^{(obs)}$ and $Y_s(\boldsymbol{\xi})$, are the observed values of the output in the training data and the predicted output for sample s, respectively. During inference, the output can be estimated using equation A.13.

$$J(c) = \frac{1}{2S}\sum_{s=1}^{S}\{Y_s^{(obs)} - Y_s(\boldsymbol{\xi})\}^2 \tag{3.4}$$

$\Phi_i(\boldsymbol{\xi})$s are constructed by only using the moments of $\boldsymbol{\xi}$ and can be constructed with limited data. No other statistical information including the underlying distributions is required. The multivariate basis functions can be obtained by taking the cross product of univariate basis functions as expressed in equation A.15.

$$\Phi_i(\boldsymbol{\xi}) = \prod_{j=1}^{N} P_j^{(\alpha_j^i)}(\xi), \tag{3.5}$$

$$\sum_{j=1}^{N} \alpha_j^i \leq M, \quad i = 1, 2..., N, \tag{3.6}$$

where α is an $M \times N$ matrix, which contains the corresponding degree for the variable index j in the expansion term k. Hence, index α_j^i enumerates all possible products of individual univariate basis functions. $P^{(k)}$ are the orthonormal univariate polynomial basis of order k as shown in equation A.17, where $p_i^{(k)}$ are their corresponding coefficients.To compute $P^{(k)}(\xi)$, $2k$ moments need to be computed.

$$P^{(k)}(\xi) = \sum_{i=0}^{k} p_i^{(k)}\xi^i, \quad k = 0, 1, ..., d. \tag{3.7}$$

By Utilizing the orthogonality of $P^{(k)}(\xi)$ and assuming that the leading coefficient $p_k^{(k)} = 1$, every $p_i^{(k)}$ can be computed from only the moments of its input Oladyshkin and Nowak (2012). The kth sample moment of the random variable ξ for S samples is expressed in equation 3.8. This allows to write the system of linear equations based on the raw moments, conveniently, in a matrix form as in equation A.27.

$$\mu_k = \frac{1}{S}\sum_{s=1}^{S}\xi_s^k \tag{3.8}$$

$$
\begin{bmatrix}
\mu_0 & \mu_1 & \dots & \mu_k \\
\mu_1 & \mu_2 & \dots & \mu_{k+1} \\
\vdots & \vdots & \vdots & \vdots \\
\mu_{k-1} & \mu_k & \dots & \mu_{2k-1} \\
0 & 0 & \dots & 1
\end{bmatrix}
\times
\begin{bmatrix}
p_0^{(k)} \\ p_1^{(k)} \\ \vdots \\ p_{k-1}^{(k)} \\ p_k^{(k)}
\end{bmatrix}
=
\begin{bmatrix}
0 \\ 0 \\ \vdots \\ 0 \\ 1
\end{bmatrix}. \tag{3.9}
$$

As a concrete example, consider a system with $\boldsymbol{\xi} = \{\xi_1, \xi_2\}$. Suppose a $2nd$ order model needs to be constructed for this system. The first step is to compute the coefficients of the $0th$ and $1st$ order polynomials for each variable as expressed in equation 3.10. These are functions of the moments (μ) of each variable. Next, the uni-variate polynomials up to the $2nd$ degree need to be constructed as expressed in equation 3.11.

$$p_0^{(2)} = -1, \; p_1^{(2)} = -\mu_3, \; p_2^{(2)} = 1 \tag{3.10}$$

$$P^{(0)}(\xi) = 1, \; P^{(1)}(\xi) = \xi, \; P^{(2)}(\xi) = \xi^2 - \mu_3\xi - 1 \tag{3.11}$$

For the $2nd$ order model, there are six multivariate polynomial that are obtained by taking the cross product of the univariate polynomials. This is shown in equation 3.12. The final step is to compute the six unknown coefficients, c_i, which are solved using least squares.

$$\Phi(\xi_1, \xi_2) = \{1, \; \xi_1, \; \xi_2, \; \xi_1\xi_2, \xi_1^2 - \mu_3^1\xi_1 - 1, \; \xi_2^2 - \mu_3^2\xi_2 - 1\} \tag{3.12}$$

3.4.1 Quantifying Uncertainty with aPC

The characteristic statistical quantities of $Y(\boldsymbol{\xi})$, which are its mean (μ) and variance (σ^2), can be evaluated directly from the coefficients c_i. This is expressed in

equation A.14.

$$\mu_Y = c_0, \quad \text{and} \quad \sigma_Y^2 = \sum_{i=1}^{M} c_i^2. \tag{3.13}$$

The coefficients are computed as a one time cost and remain unchanged during inference, hence the estimated mean and variance. To detect an out-of-distribution observation and estimate the uncertainty, the predicted outcome is compared against the mean and variance of the model $Y(\boldsymbol{\xi})$. To see how far, the predicted output is from the mean of $Y(\boldsymbol{\xi})$, in terms of standard deviation.

3.4.2 Global Sensitivity Analysis with aPC

The Sobol indices Saltelli and Sobol' (1995); Sobolá (2001); Archer *et al.* (1997) have gained more attention as opposed to other measures due to its higher accuracy in most models. Previous studies introduced PCE models that allow the computation of Sobol indices analytically as a post-processing of the PCE coefficients Sudret (2008). Hence, the cost of computing the sensitivity indices is reduced to estimating the PCE coefficients. This applies to the aPC method as well.

To investigate the influence of all the input parameters on the model output, the Sobol indices are computed as a measure of sensitivity estimation using equation 3.14.

$$S_{i_1,i_2,...,i_s} = \frac{\sum_{j=0}^{M-1} \chi_j c_j^2}{\sum_{j=1}^{M-1} c_j^2} \tag{3.14}$$

$$\chi_j = \begin{cases} 1, & if \alpha_j^k > 0, \forall j \in (i_1, i_2, ..., i_s) \\ 0, & if \alpha_j^k = 0, \exists j \in (i_1, i_2, ..., i_s) \end{cases}$$

where $S_{i_1,i_2,...,i_s}$ is the Sobol index that indicates what fraction of the total variance of PE can be attributed to the contributions of the input parameters $i_1, i_2, ..., i_s$, jointly

and individually.

The total Sobol index (S_j^T) expresses the total contribution to the variance of model output PE due to the uncertainty of an individual input parameter in all cross-combinations with other parameters.

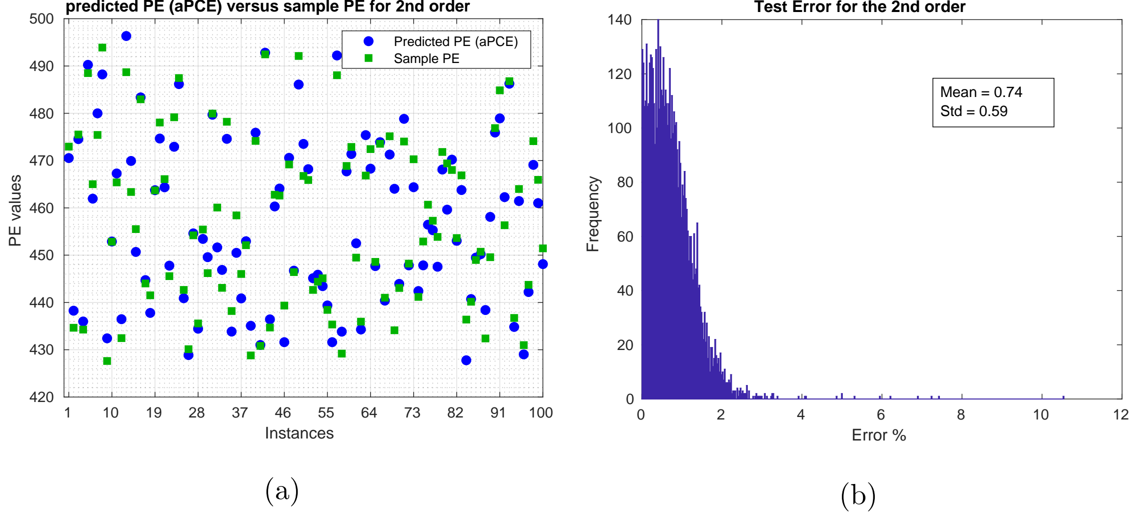
$$S_j^T = \sum_{(i_1,i_2,\ldots,i_s):j\in(i_1,i_2,\ldots,i_s)} S_{i_1,i_2,\ldots,i_s} \tag{3.15}$$

where S_j^T sums up all Sobol indices in which an input parameter appears both as univariate and joint influences.

3.5 Motivation

In this section, the application of this framework is demonstrated on a real-world regression dataset and its point-wise prediction accuracy is compared against two common methods, namely, support vector machine (SVM) and NN. The regression dataset is referred to as the combined cycle power plant (CCPP) Tfekci (2014); Kaya and Tufekci (2012); Lichman *et al.* (2013).

This dataset contains 9568 data points collected from a CCPP over 6 years. The input features consist of hourly average ambient variables temperature (T), ambient pressure (P), relative humidity (H) and exhaust vacuum (V) and the goal is to predict the net hourly electrical energy output plant, PE in short. The input uncertainty in this example comes from the noise in the sensors and data measurements. Hence, $\boldsymbol{\xi} = \{T, V, P, H\}$. The output PE is represented by the polynomial expansion as expressed in Equation 3.16 and 3.17. For this data set, $N = 4$ and the polynomials are expanded up to the 2nd order, i.e. $d = 2$. Hence, there are 15 polynomial terms, i.e. $M = 15$.



(a) (b)

Figure 3.4: (a) Sample PE Versus the Predicted PE with aPC for the First 100 Instances. (b) The Frequency Histogram of the Test Error (%) for the 2nd Order Expansion. The Maximum Error is Around 16 Standard Deviations Far From the Mean.

$$PE(T,V,P,H) = \sum_{i=1}^{15} c_i \Phi_i(T,V,P,H), \tag{3.16}$$

$$\begin{aligned} PE(T,V,P,H) \approx c_0 + c_1(T) + c_2(V) + c_3(P) + c_4(H) + c_5(T^2 - \mu_3 T - 1) + \\ c_6(V^2 - \mu_3 V - 1) + c_7(P^2 - \mu_3 P - 1) + c_8(H^2 - \mu_3 H - 1) + \\ c_9(TV) + c_{10}(TP) + c_{11}(TH) + c_{12}(VP) + c_{13}(VH) + c_{14}(PH) \end{aligned} \tag{3.17}$$

Figure 3.4a illustrates the actual value of the PE from the dataset and the predicted value using this approach expanded up to the $2nd$ order.

Table 3.1 presents the regression errors of aPC, NN and SVM models in terms of RMSE. In this experiment, the kernels used in the SVM are polynomials and the NN model is a one layer network with 50 hidden units. The average RMSE of the aPC is the same as NN and is smaller than that of the SVM model.

Table 3.1: RMSE for the Power Plant.

	aPC	NN	SVM Tfekci (2014)
RMSE	4.02	4.02	4.8

The average point-wise prediction error as compared to the sample outputs is less than 1%, using aPC. The p.d.f of PE is also estimated with MC simulations by sufficiently sampling from $\boldsymbol{\xi}$ and evaluating the responses with the constructed aPC model. This is illustrated in Figure 3.5. The evaluation of aPC, which are essentially polynomials, is computationally inexpensive, hence the MC simulations. To measure the distance between the sample p.d.f (f_{PE}) and the estimated p.d.f ($\tilde{f}_{PE}$), a common method referred to as Kullback-Leibler divergence (KL) Kullback and Leibler (1951) (also known as relative entropy) is employed as expressed in equation 3.18.

$$D_{KL}(\tilde{f}||f) = \sum_{x \in \mathcal{X}} \tilde{f}(x) ln(\frac{\tilde{f}(x)}{f(x)}), \tag{3.18}$$

In this experiment, since the true p.d.f is not known, the sample p.d.f is considered as the ground truth. The D_{KL} value for the p.d.fs illustrated in Figure 3.5 is 0.04, which is close to 0. It should be noted that the methods employed in Tfekci (2014) cannot efficiently provide the estimated p.d.f and a measure for UQ, in terms of computation. This is evaluated more in-depth in Section 3.9.

The mean and variance of PE are also computed by evaluating equation A.14. The computed mean and variance are 453.5 and 205.1, respectively while the sample mean and variance are 454.3 and 291.2. The interval $\mu \pm \sigma$ provided by the aPC method is generally contained in the interval computed from the sample data.

Figure 3.6 illustrates the use of estimated mean and variance to detect outliers during inference.. The dashed line and band demonstrate the estimated mean with

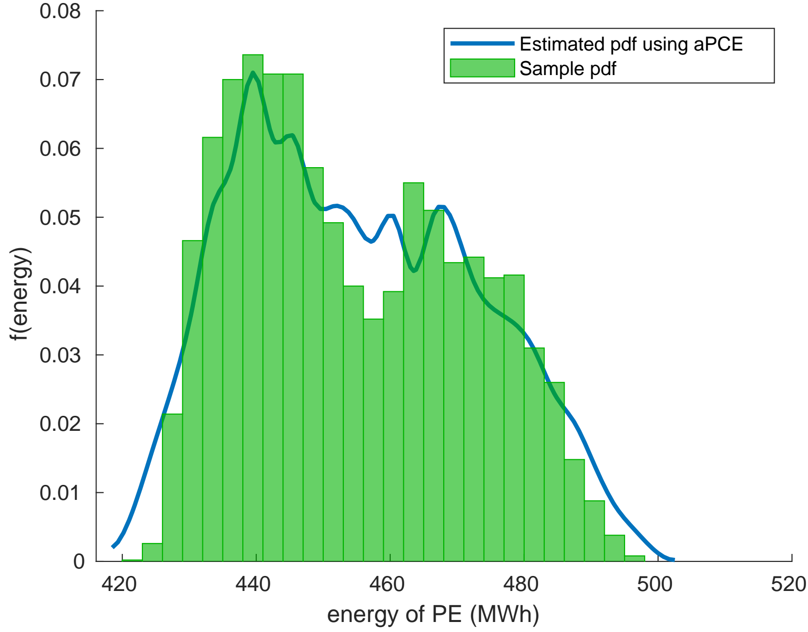


Figure 3.5: The Bars Represent the Histogram Obtained from the Sample Power Plant Energy and the Curve Shows the Estimated P.D.F of the Energy Using aPC. The KL Divergence Value (d_{Kl}) Between the Two P.D.Fs Is .04, Which Is Close to 0.

one standard deviation. In this experiment, 200 samples are generated, out of which 143 are outliers, shown with cross-shape points. The remaining data shown in circles are randomly selected from the original sample data. The outliers are generated by randomly modifying one or more of the input variables to be sampled from outside of the original sample data. In this experiment, with one standard deviation, the aPC model can detect all the outliers, while misrepresenting only 0.05% of the original samples as outliers.

An experiment is conducted to eliminate the input variables either individually or jointly based on the computed total Sobol indices. The objective is to find the largest set of input parameters that can be removed with negligible increase in the test error and change in the estimated mean and variance. As there are four input parameters in CCPP, $\{i_1, i_2, ..., i_s\}$ in equation 3.14 correspond to $\{T, V, P, H\}$, re-

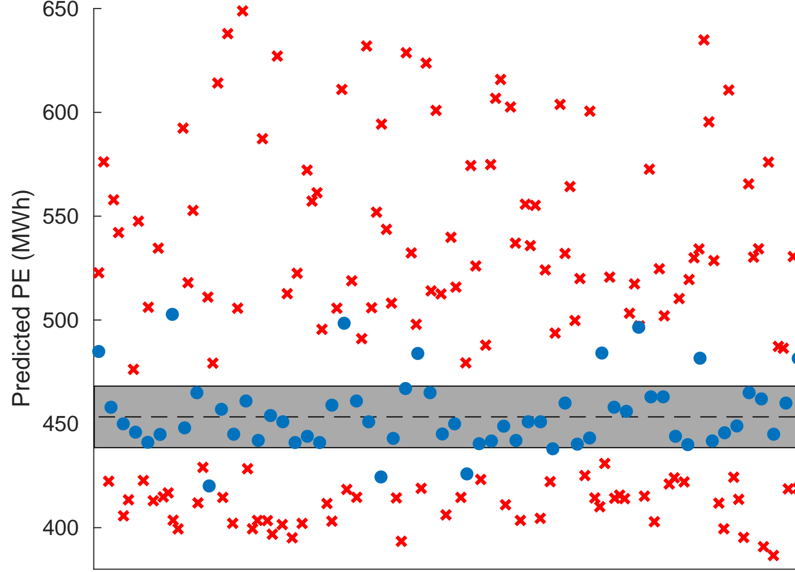


Figure 3.6: Identifying Outliers. The Dashed Line and Band Demonstrate the Estimated Mean with One Standard Deviation.

spectively. All the possible combinations of the Sobol indices are $\{S_T, S_V, S_P, S_H\}$, $\{S_{TV}, S_{TP}, S_{TH}, S_{VP}, S_{VH}, S_{PH}\}$, $\{S_{TVP}, S_{TVH}, S_{TPH}, S_{VPH}\}$ and $\{S_{TVPH}\}$. The total Sobol indices for the input parameters $\{T, V, P, H\}$ are expressed in Equations 3.19 - 3.22.

$$S_T^T = S_T + S_{TV} + S_{TP} + S_{TH} + S_{TVP} + S_{TVH} + S_{TPH} + S_{TVPH} = 0.902 \tag{3.19}$$

$$S_V^T = S_V + S_{TV} + S_{VP} + S_{VH} + S_{TVP} + S_{TVH} + S_{VPH} + S_{TVPH} = 0.070 \tag{3.20}$$

$$S_P^T = S_P + S_{TP} + S_{VP} + S_{PH} + S_{TVP} + S_{TPH} + S_{VPH} + S_{TVPH} = 0.008 \tag{3.21}$$

$$S_H^T = S_H + S_{TH} + S_{VH} + S_{PH} + S_{TVH} + S_{TPH} + S_{VPH} + S_{TVPH} = 0.020 \tag{3.22}$$

Hence, the constructed model ranks the influence of the inputs on the variance of PE as $\{P, H, V, T\}$, from lowest to highest. For example, in this dataset, P, which has the lowest Sobol index can be eliminated with negligible impact on the accuracy of the model. This increases the test error, estimated mean and variance, by 0.04%,

0.02%, and 1.4%, respectively.

3.6 Incremental Computations to Construct Higher Order Models

This section demonstrates that a constructed model for expansion order d can be used to construct higher order models (e.g. $d+1$). This is shown by incrementally adding to the computation instead of recalculating everything. This allows a much faster training time.

Let d be the order of expansion that the model is first constructed, and $\boldsymbol{\xi} = \{\xi_1, \xi_2, ..., \xi_N\}$ be the set of N input random variables. As shown in equation A.13, $Y^{(d)}(\boldsymbol{\xi})$ has $M = \binom{N+d}{d}$ coefficients and multi-variate polynomial terms that include all the polynomials from degree 0 to d.

Suppose the accuracy of $Y^{(d)}(\boldsymbol{\xi})$ is not sufficient and a higher order model needs to be constructed which is expressed in equation 3.23.

$$Y^{(d+1)}(\boldsymbol{\xi}) = \sum_{i=1}^{\binom{N+d+1}{d+1}} c_i' \Phi_i'(\boldsymbol{\xi}), \tag{3.23}$$

$\Phi'(\boldsymbol{\xi})$ consists of two concatenated vectors, i.e. $[\Phi(\boldsymbol{\xi})|\Phi^*(\boldsymbol{\xi})]$, in which $\Phi^*(\boldsymbol{\xi})$ is the additional multi-variate polynomials that need to be computed for order $d+1$. As there are $\binom{N+d+1}{d+1}$ terms in $\Phi'(\boldsymbol{\xi})$ and M of which are in $\Phi(\boldsymbol{\xi})$, there are $\binom{N+d}{d+1}$ terms in $\Phi^*(\boldsymbol{\xi})$).

As a concrete example, consider a system with two random variables, with $N = 2$ and $\boldsymbol{\xi} = \{\xi_1, \xi_2\}$. Suppose $Y^{(2)}(\boldsymbol{\xi})$ needs to be constructed from its preceding model $Y^{(1)}(\boldsymbol{\xi})$. $\Phi'(\boldsymbol{\xi})$ in $Y^{(2)}(\boldsymbol{\xi})$ has 6 terms and $\Phi(\boldsymbol{\xi})$ in $Y^{(1)}(\boldsymbol{\xi})$ has 3 polynomial terms. Since the multi-variate polynomials are orthogonal to each other, to construct Φ', the 3 multi-variate polynomials terms in Φ can be reused and an additional 3 terms in $\Phi^*(\boldsymbol{\xi})$ need to be computed. This is shown as a vector in equation 3.24, where $\mu_i^{(j)}$ is

the ith moment of the jth variable.

$$\Phi'(\boldsymbol{\xi}) = \Bigg[\underbrace{1,\ \xi_0,\ \xi_1}_{\Phi(\boldsymbol{\xi})}\Bigg|\ \underbrace{\xi_0^2 - \mu_3^{(0)}\xi_0 - 1,\ \xi_0\xi_1,\ \xi_1^2 - \mu_3^{(1)}\xi_1 - 1}_{\Phi^*(\boldsymbol{\xi})}\Bigg] \tag{3.24}$$

The goal of the aPC model is to construct the polynomials in equations A.15 and A.17 to form an orthonormal basis for arbitrary distributions. That being said, let us briefly describe the orthogonal decomposition theorem. This theorem states that a vector y in $\mathbb{R}^n$ can be written uniquely as in equation 3.25.

$$y = \hat{y} + z, \tag{3.25}$$

where $\hat{y}$ is in W, a subspace of $\mathbb{R}^n$, and z is in $W^{\perp}$. In fact, if $\{u_1, ..., u_p\}$ is any orthogonal basis of W, then $\hat{y}$ and z are as in equations 3.26 and 3.27.

$$\hat{y} = \frac{y.u_1}{u_1.u_1}u_1 + ... + \frac{y.u_p}{u_p.u_p}u_p, \tag{3.26}$$

$$z = y - \hat{y} \tag{3.27}$$

Vector $\hat{y}$ is the orthogonal projection of y onto W as illustrated in figure 3.7.

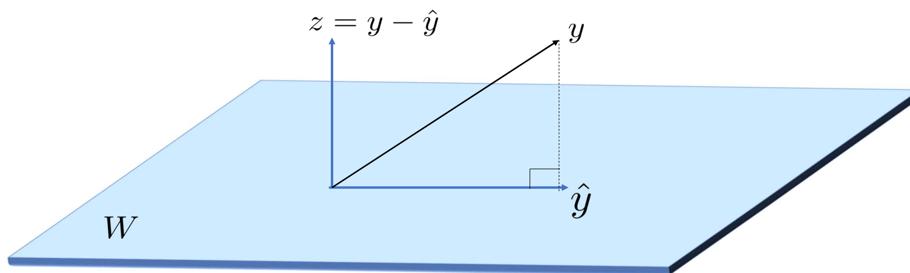


Figure 3.7: The Orthogonal Projection of y into W.

It is shown in Oladyshkin and Nowak (2012) that any polynomial of degree k in the aPC model is orthogonal to all lower-order polynomials. This is true for both

uni-variate polynomial $P^{(k)}$ and multi-variate polynomial $\Phi^{(k)}$. Consequently, polynomials of degree $d+1$ and d are orthogonal to each other and belong to subspace $W^{\perp}$ and W, respectively. That said, $Y^{(d+1)}(\boldsymbol{\xi})$ can be written as a linear combination of $Y^{(d)}(\boldsymbol{\xi})$ and the additional polynomial terms as shown in the orthogonal decomposition theorem. This is expressed in equation 3.28. This concludes that to construct a higher order model (e.g. $d+1$), the set of polynomials terms that form the model for only degree $d+1$ need to be additionally computed. The rest of the polynomials can be directly taken from the precomputed model for order d.

$$Y^{(d+1)}(\boldsymbol{\xi}) = \underbrace{\sum_{i=1}^{M} c_i \Phi_i(\boldsymbol{\xi})}_{\hat{y}} + \underbrace{\sum_{j=M+1}^{\binom{N+d+1}{d+1}} c'_j \Phi^*_j(\boldsymbol{\xi})}_{z} \tag{3.28}$$

$$Y^{(d+1)}(\boldsymbol{\xi}) = Y^{(d)}(\boldsymbol{\xi}) + \sum_{j=M+1}^{\binom{N+d+1}{d+1}} c'_j \Phi^*_j(\boldsymbol{\xi}) \tag{3.29}$$

To demonstrate the efficiency of the incremental computations, empirically, the aPC model for the CCPP dataset is constructed for various order of expansions. Figure 3.8 illustrates the speed-up that is achieved by incrementally adding to the computations as opposed to recomputing all the polynomial terms from order 1. For example, for $d = 5$, the execution time of the aPC model constructed from the $1st$ order is $2.57X$ larger than the model only constructed the additional terms from order 4 to 5.

3.7 Extension to aPC for Classification Tasks

In the previous sections, the aPC method is discussed for regression tasks where the outputs have continuous values and are solved using least squares as in Equation 3.4. This cost function cannot be employed for the classification tasks as the

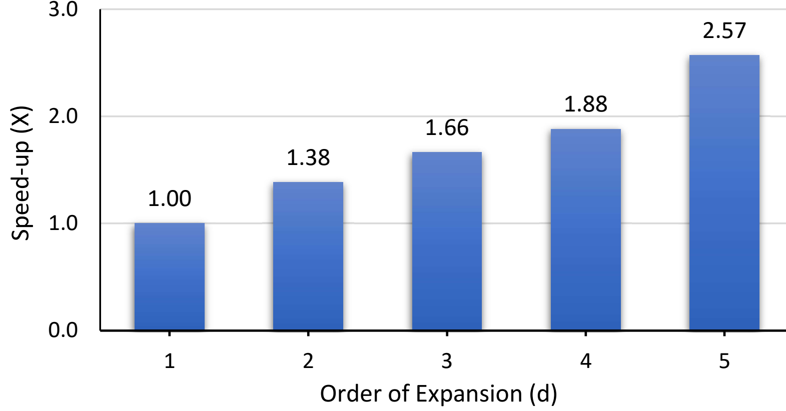


Figure 3.8: The Speed up Achieved by Incrementally Adding to the Computation. This Is Shown up to the $5th$ Order. Greater Speed-up Is Achieved for Larger d.

outputs are bounded and discrete. Hence, we employ the loss functions used in logistic regression in machine learning methods to extend the applicability of this approach to tasks with discrete outputs. The cost function $J(c)$ for binary classification, also used in logistic regression, is presented in Equation 3.30.

$$J(c) = -\frac{1}{m}[\sum_{s=1}^{S} Y_s^{(obs)} log(Y_s(\boldsymbol{\xi})) + (1 - Y_s^{(obs)}) log(1 - Y_s(\boldsymbol{\xi}))], \tag{3.30}$$

where $Y_s(\boldsymbol{\xi})$ is the predicted output for sample s using aPC, $Y_s^{(obs)}$ is its observed output and c is the vector of model coefficients. The objective is to minimize the cost function J with respect to c using algorithms such as gradient descent. For multi-class classification, where the target outputs belong to more than two categories, the *one-vs-all* method is used. Generally in the *one-vs-all* method, the classifier $Y^i(\boldsymbol{\xi})$ is trained for each class i to predict the probability that $Y^i(\boldsymbol{\xi}) = i$. During inference, given a new input, the output class is the one that maximizes $Y^i(\boldsymbol{\xi})$.

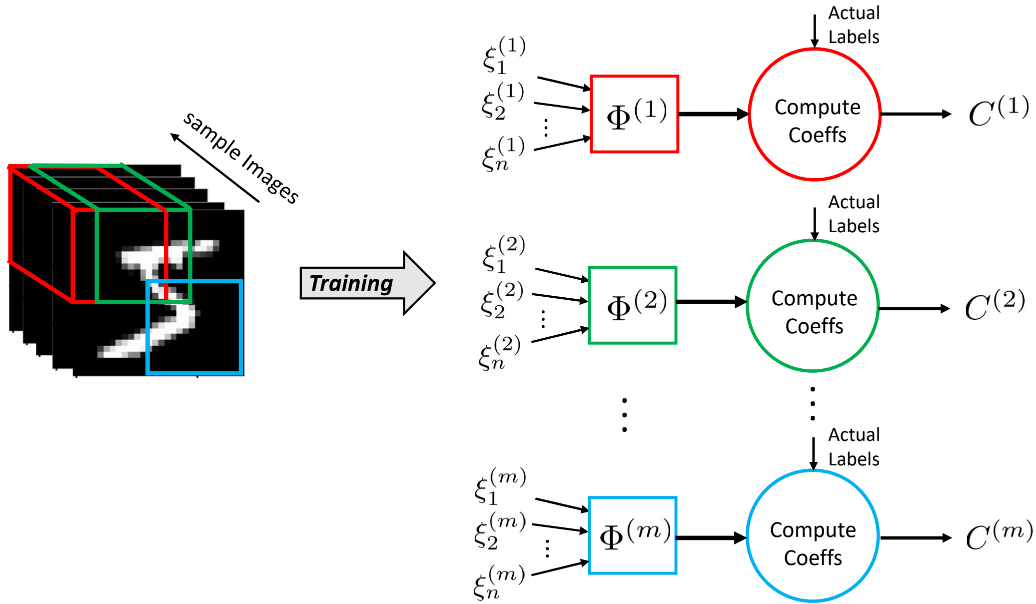


Figure 3.9: The Proposed *Training* Procedure for Solving the Scalability Issue and Applying aPC on Handwritten Digit Recognition. The Number of Partitions and the Number of Pixels per Partition Are Denoted as m and n, Respectively. The Output of This Procedure Is m Tuned Coefficients (C) Which Will Be Used for the Inference Phase.

3.7.1 Address the Scalability Issue in Classification Tasks

One limitation of aPC is the difficulty in model construction with large number of input parameters. This limitation leads to either a substantially high computational cost or becomes infeasible to model. For example, consider constructing an aPC model for the well-known handwritten digit dataset, referred to as MNIST LeCun and Cortes (2010). The inputs are handwritten digit images of size 28×28 and the objective is to detect the digits and assign them to their corresponding class labels. The overall structure of this application is illustrated in Figure 3.11, in which the computational model is typically replaced with a well-known NN model, referred to as LeNet5 Lecun *et al.* (1998).

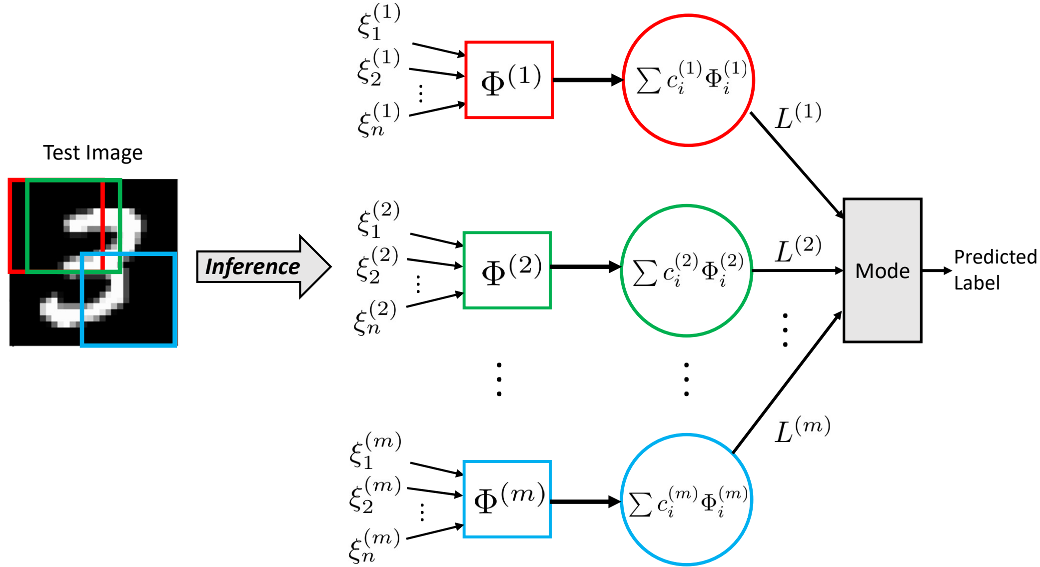


Figure 3.10: The Proposed *Inference* Procedure for Predicting the Digit Label of a given Test Image Using the Coefficients Computed in the Training Procedure.

The LeNet5 network consists of two convolutional and two pooling layers which are followed by two fully connected layers and a classifier (i.e. softmax) as shown in Figure 3.11. In the handwritten digit problem, the output of the softmax layer is a 10 dimensional vector of probabilities, in which the index of the highest probability is selected as the output digit.

In real-world examples, the input images are typically noisy and the input pixels belong to a stochastic domain. Hence, we consider all the input pixels (i.e. 784) to be random variables. For the 1st order expansion, $\binom{784+1}{1} = 785$ polynomial terms need to be constructed which is feasible. However, the first order is often not accurate. Increasing the order of expansion increases the number of polynomial terms exponentially. For example, for the 2nd and 3rd orders, around $308k$ and 80 million polynomial terms need to be constructed, respectively. This incurs high computational cost and memory footprint and is infeasible. Note that in this example, the input images are smaller than many other applications which involve larger number of input variables,

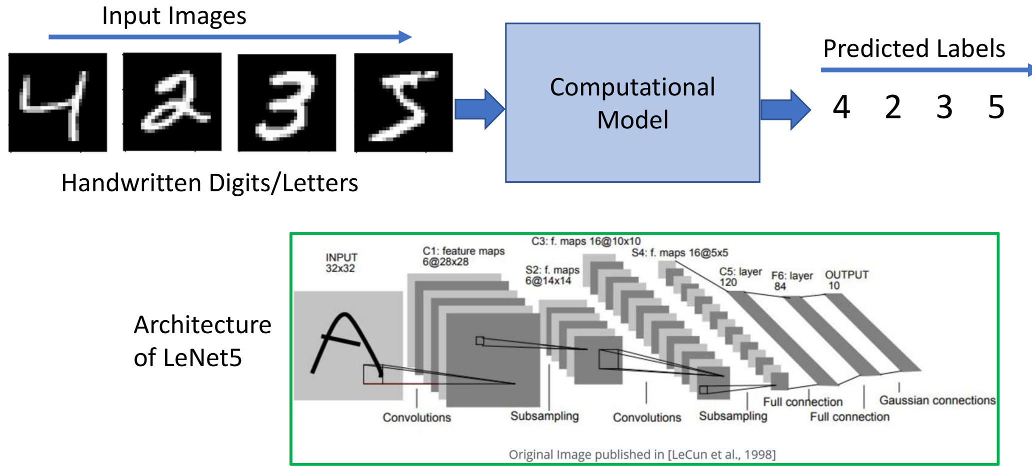


Figure 3.11: The Overall Structure of the Handwritten Digit/Letter Detection Problem for the MNIST Dataset. Typically the Computational Model Is Replaced with LeNet5 as It Has Demonstrated a High Accuracy Of 99.9%.

so the scalability issue holds for those applications as well.

To address this issue, this section proposes to partition the input images into smaller groups of random variables with overlapping regions. The overall *training* procedure is depicted in Figure 3.9. Similar to the sliding windows in the convolutional neural networks, this method uses a fixed-size sliding window with a stride. The training images are partitioned into smaller blocks. In Figure 3.9, the red, green and blue boxes are shown as examples of such blocks .

The output of this step is m partitions with n pixels in each partition, denoted as ξ_j^k, where $k \in [1, 2, ..., m]$ and $j \in [1, 2, ..., n]$. The next step is to compute the sample moments of all the given training images for a given partition and construct the multi-variate polynomials. This is denoted as Φ^k. Given Φ^k and the actual class labels in the training dataset, the set of coefficients for each partition, denoted as $C^{(k)}$, are trained using the *one-vs-all* method.

Figure 3.10 illustrates the proposed *inference* procedure which is used to predict

the digit label of a given test image. The partitioning on a given test image is performed the same as in the training phase. After evaluating the Φ^k for the test image, the prediction is performed by simply evaluating the expression in Equation 3.31 for each partition.

$$L^k = \sum_{i=1}^{M} c_i^{(k)} \Phi_i^{(k)}, \tag{3.31}$$

where L is the predicted label. k and M denote the partition and number of polynomial terms, respectively. The output of this step is an m dimensional vector consisting of the labels predicted by all the m partitions. The mode (i.e. highest count) of that vector is computed at the end to report the final digit label. As a concrete example, let $m = 10$ and the output vector $L = [2, 5, 2, 2, 5, 7, 1, 2, 2, 5]$ for a given test image. Digit Label 2 appeared the most in the predicted output vector and hence is reported as the final output.

3.8 A Computation Graph for aPC

This section turns the aPC method into an executable computation graph. In addition, it explains the operations that can be parallelized for even faster execution time. Figure 3.12 depicts the computation graph of aPC for training. The ovals determine the inputs and outputs. The input to this graph is the normalized sample data with zero mean and unit variance, $\boldsymbol{\xi}$, given N and S as the number of input variables and training samples, respectively. Each node in rectangular shape represents an operation with its output size presented below it. Every edge represents the data generated by its source node. For example, the first node computes the sample moments of every input variable as expressed in equation 3.8. The number of moments that need to be computed for every input is twice the order of expansion d,

hence the dimension $2d \times 1$. Since there are N random variables, there are N edges with each representing the moments vector for its corresponding input. Computing the coefficients, C, enables global sensitivity analysis (GSA) (see sections 2.9 and 3.5), UQ and training error computation. If the training error is high, a higher order model $Y^{(d+1)}(\boldsymbol{\xi})$ can be constructed with incremental computations as discussed in section 3.6. The structure of this graph intuitively demonstrates the sequential and parallel operations. All the nodes at one level can be executed simultaneously, while the tasks from top to bottom are data dependent and are run in sequence.

3.9 Experimental Results

The proposed data-driven framework is evaluated on a range of tasks and datasets. For regression, similar to the prior works Gal and Ghahramani (2016); Lakshminarayanan *et al.* (2017), the regression benchmarks Hernández-Lobato and Adams (2015) are considered. For classification, the framework is evaluated on MNIST LeCun and Cortes (2010) and CIFAR-10 Krizhevsky (2009). In addition, the out-of-distribution task for classification is addressed on MNIST/NotMNIST Lakshminarayanan *et al.* (2017).

3.9.1 Regression Experiments

In these set of experiments for regression benchmarks, the proposed framework is compared against the state-of-the-art methods for predictive uncertainty estimation using neural networks. The same experimental setup as in Lakshminarayanan *et al.* (2017); Gal and Ghahramani (2016) is used. For each dataset, a neural network with one hidden layer consisting of 50 hidden units are constructed except the protein dataset which is larger and contains 100 hidden units. Each dataset is split into 20 train-test folds except the protein dataset which uses 5 folds. The results collected

by the proposed framework are compared against MC dropout Gal and Ghahramani (2016) and deep ensembles Lakshminarayanan *et al.* (2017) and reported in Table 3.2. For the deep ensembles approach, five networks are used as reported in Lakshminarayanan *et al.* (2017). Our method outperforms or is competitive with existing methods in terms of RMSE.

Figure 3.13 depicts the normalized train time of our method along with MC dropout and deep ensembles on the regression benchmarks. As compared to deep ensembles, our method is orders of magnitude faster. As compared to MC dropout, our method is on average $54X$ faster with the maximum speed up of $210X$.

Figure 3.14 illustrates the fraction of train data used for training deep ensembles, MC dropout and our method. To achieve the RMSE results reported in Table 3.2, both deep ensembles and MC dropout need to use all the available training data. However, our method requires limited train data to compute the coefficients of the polynomials.

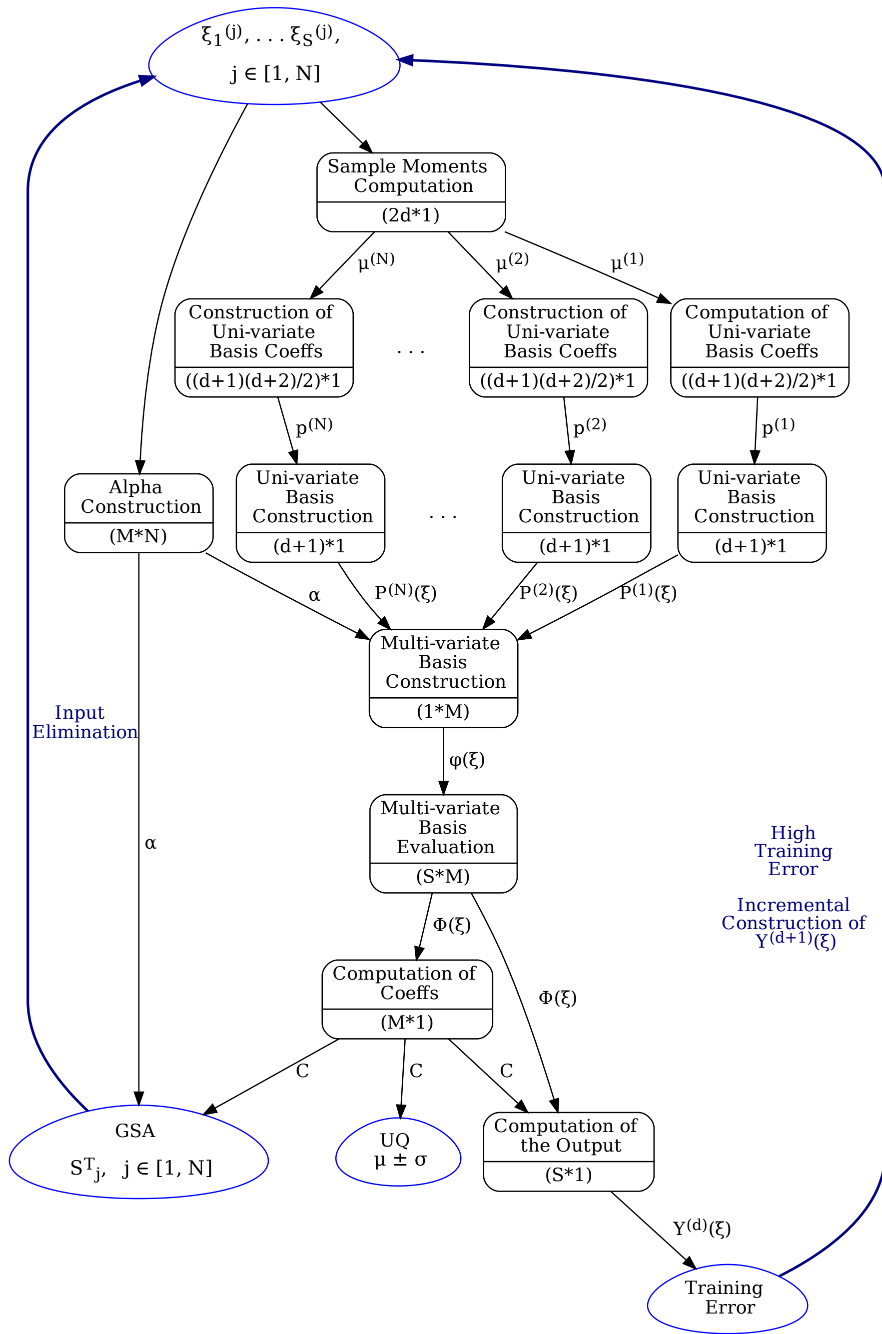


Figure 3.12: Computation Graph of aPC for Training. The Inference Graph Has the Same Operations Except the Computation of the Coeffs Node. Hence It Is Omitted for Brevity.

Table 3.2: Comparative Results on Regression Benchmarks.

	RMSE		
Datasets	MC Dropout Gal and Ghahramani (2016)	Deep Ensembles Lakshminarayanan *et al.* (2017)	This Work
Boston Housing	2.97±0.85	3.92±1.01	**2.91± 0.67**
Concrete Strength	5.23±0.53	6.03±0.47	**5.22± 0.46**
Energy Efficiency	1.66± 0.19	2.86±0.29	**1.50±0.27**
Kin8nm	0.10±0.00	0.09±0.00	**0.09±0.00**
Naval Propulsion Plant	0.01±0.00	0.00±0.00	**0.00±0.00**
Power Plant	4.02±0.18	4.09±0.16	**4.02±0.16**
Protein Structure	4.36±0.04	4.71±0.06	**4.35±0.04**
Wine Quality Red	**0.62±0.04**	0.64±0.04	0.63±0.03
Yacht Hydrodynamics	**1.11±0.38**	1.58±0.46	1.17±0.32

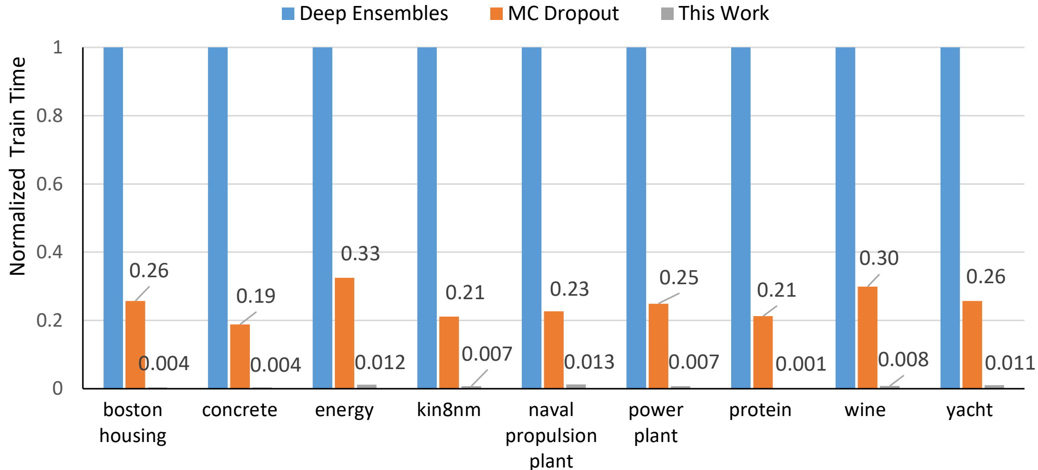


Figure 3.13: The Normalized Train Time of This Work along with MC Dropout and Deep Ensembles on the Regression Benchmarks.

3.9.2 Classification Experiments

For the classification tasks, the functionality of this work is demonstrated on the well-known image classification datasets, MNIST LeCun and Cortes (2010) and CIFAR-10 Krizhevsky (2009). In MNIST, the inputs images of size 28×28 and the objective is to detect the digits and assign them to their corresponding class labels. The well-known NN model for this classification task is LeNet5 Lecun *et al.* (1998). In CIFAR-10, there are 10 classes and the images are of size 32×32.

We aim to replace the computational model (i.e. LeNet5), with our proposed framework and compare its accuracy and performance. The main reason is due to the fact that in real-world examples, the input images are typically noisy and the input pixels belong to a stochastic domain. Since there is uncertainty in the inputs (e.g. noise), it is important to estimate the response statistics as explained in the preceding sections. Moreover, unlike NN models, this method does not require hyper-parameter tuning such as deciding on the number of layers, number of hidden nodes,

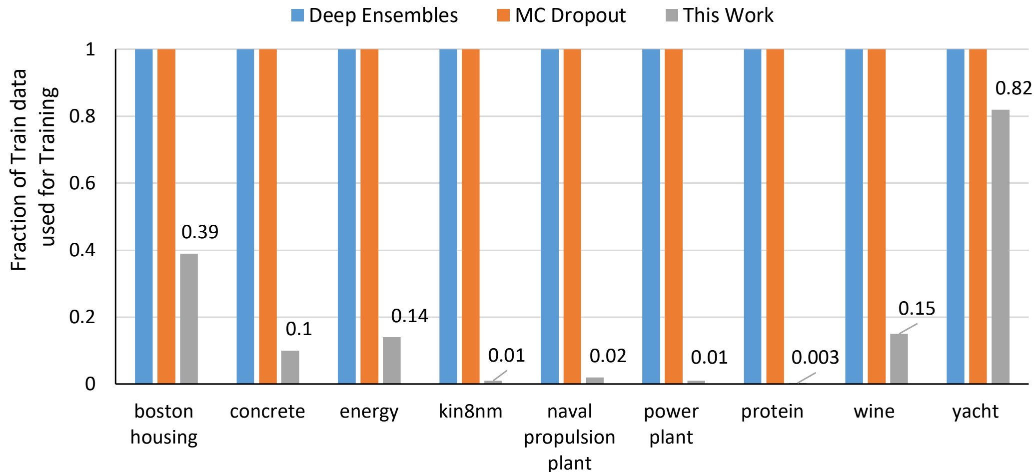


Figure 3.14: The Fraction of Train Data Used for Training Deep Ensembles, MC Dropout and This Work on the Regression Benchmarks.

etc and is expected to be computationally less expensive for uncertainty propagation as opposed to its equivalent weight space sampling methods or ensemble methods.

For the classification task, the performance of our approach is evaluated on MNIST datasets LeCun and Cortes (2010), which consists of 60k images in the train set and 10k in the test set. The images are of size 28×28. The neural network used for deep ensembles and MC dropout is LeNet5 Lecun *et al.* (1998). The accuracy of these methods are reported in Table 3.3. In terms of final accuracy, our method achieves comparable results as compared to MC dropout and deep ensembles. The reported execution times are normalized CPU times.

Figure 3.15 and 3.16 compare the execution time and the amount of data used for training in this work and the other two existing methods. This work runs $7.1X$ and $5.7X$ faster for MNIST and CIFAR10, as compared to the fastest of the other models, i.e. deep ensembles. The amount of data required for this work is 33% and 40% of the other models, for MNIST and CIFAR10, respectively. This work takes longer to train as compared to MC Dropout. Since this network is using only one

Table 3.3: Comparative Results on Image Classification.

	Accuracy %	
Method	MNIST	CIFAR-10
Deep Ensembles	98.88	95.6
MC Dropout	98.19	95.2
This Work	98.30	94.9

layer of neurons, typically, it requires more neurons to be able to reach the same level of accuracy.

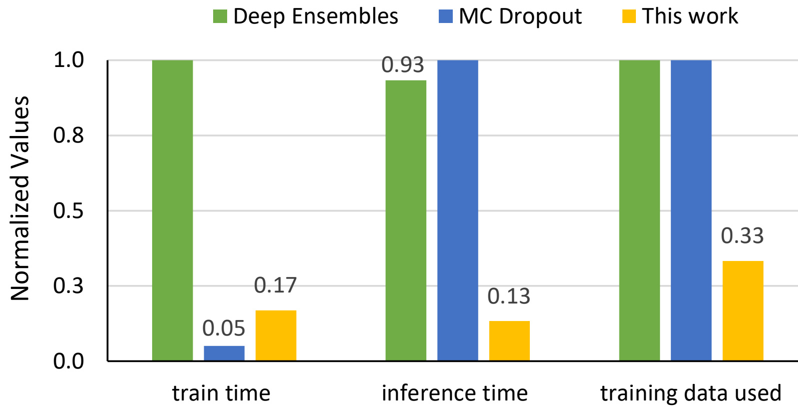


Figure 3.15: Comparison Between the Execution Time and the Amount of Data Used for Training in This Work and the Other Two Existing Methods For MNIST.

3.9.3 Uncertainty Evaluation: Test Examples from Known vs Unknown Classes

The uncertainty is evaluated on out-of-distribution test samples. In real-world applications, when the test samples are not similar to the training datasets, the model must show higher uncertainty. The model is first trained on MNIST dataset and then tested on NotMNIST datasets. Figure 3.17 illustrates the percentage of the NotMNIST images that are identified correctly as outliers, given various intervals.

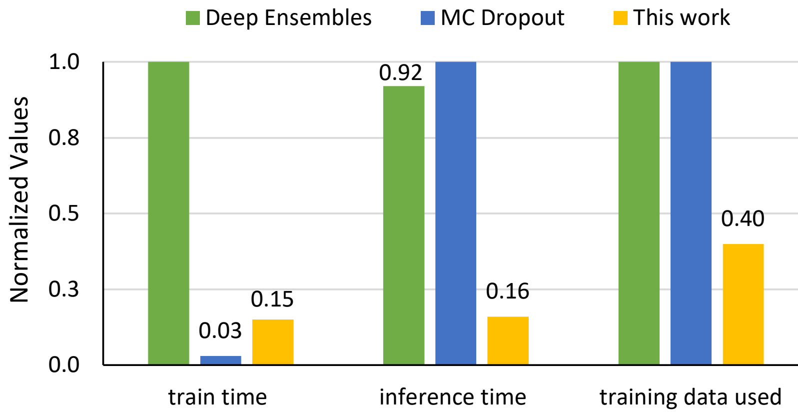


Figure 3.16: Comparison Between the Execution Time and the Amount of Data Used for Training in This Work and the Other Two Existing Methods For CIFAR-10.

The μ and σ are computed from the original model trained on the MNIST dataset. As the interval becomes smaller, the number of correct detections increase.

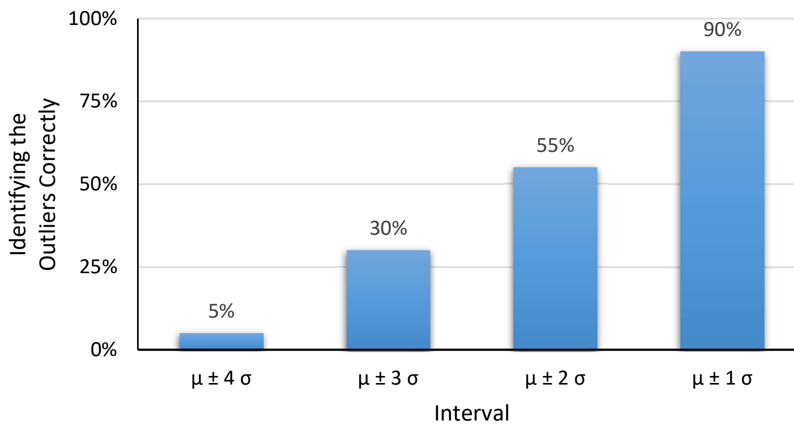


Figure 3.17: The Percentage of the NotMNIST Images That Are Identified Correctly as Outliers, given Various Intervals.

3.10 Chapter Summary

Uncertainty quantification (UQ) has gained an increasing attention in many real-world applications. UQ occurs in systems with limited knowledge about its properties

where the inputs and hence the system's output are uncertain. This work develops a robust framework for constructing data driven models for regression tasks. The surrogate model that this work employs in the presence of input variability is referred to as arbitrary polynomial chaos expansion. In addition to the point-wise prediction, this framework can provide a measure for quantifying the uncertainty in the output. This methodology is then extended to address the scalability issue in tasks with large number of input variables. Moreover, it is extended for discrete outputs so that it can be applied to classification tasks similar to neural network models. The accuracy of this framework can be improved by constructing a higher order model. This is shown by incrementally adding to the computation instead of recalculating all the polynomial basis and the coefficients. This allows a much faster training. This framework is evaluated on wide range of tasks including the regression and classification benchmarks. This work achieves competitive results as compared to the state-of-the-art network models in terms point-wise prediction. Moreover, this framework provides additional properties, such as faster training with no parameter tuning, the need for fewer training data to achieve the same level of accuracy, and the sensitivity of the output with respect to the inputs that can help with reducing the model complexity by eliminating the least significant inputs.

Chapter 4

USE CASE: POST-FABRICATION WEIGHT TUNING IN A BINARY PERCEPTRON

4.1 Problem Background

Manufacturing variations in integrated circuits have become a significant concern in nanoscale chip design, particularly at smaller process nodes Yu *et al.* (2014); Zhang *et al.* (2015). At such nodes, ignoring intra-die variations, as is often done in corner-based analysis, leads to highly pessimistic estimates of performance variables. In general, a circuit under consideration is given in some form that represents an implicit relation $F(X, Y, \xi)$, where X, Y and ξ are the set of inputs, set of outputs and a set of parameters that are subject to manufacturing variations. Each of these quantities are assumed to be vectors of arbitrary dimensions.

Modeling intra-die variations means that each statistically varying parameter within each component on a die must be represented by a distinct random variable. This often results in an extremely large space of random variables. Each point in that space represents a specific value of all the parameters, i.e., it corresponds to an instance of the circuit. The implicit relation $F(X, Y, \xi)$ is often given in the form of a large number of equations. Simulation means that the equations have to be solved to determine the response Y for a given input X and given values of ξ. To obtain a distribution of a circuit's response over the space of possible manufactured outcomes, a large sample of the parameter space is generated and each corresponding circuit instance is simulated. Sample moments and the empirical distribution function are then computed from the simulated responses. This standard approach, referred

to as *Monte Carlo* simulation (MCS), becomes computationally prohibitive for most practical circuits when considering intra-die and inter-variations.

An alternate approach is to replace the implicit relation $F(X, Y, \xi)$ with an explicit function $\widetilde{Y}(X, \xi)$ that *approximates* the circuit's response. Then instead of performing MCS based on $F(X, Y, \xi)$, large number of samples of ξ would be generated and $\widetilde{Y}(X, \xi)$ would be computed. The use of $\widetilde{Y}(X, \xi)$ as a proxy for $F(X, Y, \xi)$ has been shown to result in several order of magnitude speed up over MCS. The problem of constructing $\widetilde{Y}(X, \xi)$ is not very different from how *data-driven* models are constructed in the deterministic case. Given data pairs $\{(x_i, y_i), i = 1, 2, \cdots n\}$, which are assumed to be related by some unknown function, i.e. $y = f(x)$, the task is to select a function f from some space of continuous functions which will yield a good approximation of y_i given the x_i (*supervised learning*), and which can be used to predict values of y given values of x outside the given sample (*inference*). For this problem to be well defined and tractable, the space of functions has to be restricted to a given class (e.g. linear), elements of which are determined by a set of parameters (e.g. coefficients). The model construction problem then becomes a problem of determining the optimal set of parameters from the given data set (*training*).

In the stochastic case, the approach is somewhat similar. The random variables Y (assume finite variance) can be expressed as an infinite series of multi-variate orthogonal polynomials of all (infinite) orders, referred to as *polynomial chaos* (PC) expansion Ghanem and Spanos (1991); Soize and Ghanem (2004); Augustin *et al.* (2008). The meaning of such an expansion is that the mean and variance of the expansion converge to the mean and variance of Y in limit. The (optimal) choice of the orthogonal polynomial depends on the distribution of Y. Given a polynomial basis, the coefficients of the series expansion are estimated using one of several methods Ghanem and Spanos (1991); Babuška *et al.* (2010); Zhang *et al.* (2013). The use

of PC expansions to model manufacturing variations in microelectronic circuits have demonstrated substantial speedups, often one to three orders of magnitude, when compared to MCS Bhardwaj *et al.* (2008); Goel *et al.* (2009); Ghanta and Vrudhula (2007); Vrudhula *et al.* (2006); Zhang *et al.* (2013, 2015); Ince *et al.* (2017). However, due to the limited availability of data, lack of knowledge about the underlying physical phenomena or simply mathematical convenience, these distributions of the individual random variables are often assumed to be Gaussian, in which case the optimal bases are Hermite polynomials.

The drawback of having to determine or assume a distribution function of the random variables was eliminated by the method proposed by Oladyshkin et al. Oladyshkin and Nowak (2012). In the method, referred to as *arbitrary polynomial chaos* (aPC), the coefficients of the basis polynomials are given as explicit functions of the moments of the random variables. In practice, sample moments are used, making the method completely data-driven. This eliminates the need to assume any particular distribution and also eliminates the computationally intensive task of solving integrals for determining the coefficients.

4.2 Problem Statement and Novelty

The use of aPC in microelectronics circuits to model the impact of manufacturing variations has primarily been aimed at the analysis and estimation of circuit yield prior to fabrication. In this paper, we demonstrate another novel application of aPC – namely, to *tune* parameters of individual instances of manufactured circuits to correct *failures* and consequently maximize yield. While this scenario is quite general, we demonstrate this approach on a recently reported mixed-signal circuit, referred to as *Flash Threshold Logic* (FTL). This circuit performs the function of a *binary neuron* Wagle *et al.* (2019) – i.e., computes a threshold function which is

the basic computation involved in binary neural networks Courbariaux and Bengio (2016).

The FTL circuit in Wagle *et al.* (2019) is a mixed-signal circuit – its inputs and outputs are binary, but it computes a threshold function in the analog domain. In an FTL, the weights defining a threshold function are realized by threshold voltages of flash transistors, which can be tuned after fabrication. The flash transistors effectively control the conductivity of two input networks, and when the conductivity of one network exceeds that of the other, the output is a logic one, otherwise it is a logic zero. An FTL fails when the conductivities of the two networks are too close for the sense amplifier to distinguish the difference or if they are reversed. Both are possible due to manufacturing variations. The basic concept here is that when an FTL cell fails, the threshold voltages of the flash transistors can be tuned to compensate for the variations in the other devices to fix the failure. There are two main challenges to address. First, the number of times a flash transistor's threshold voltage can be modified (i.e. number of write cycles) is limited. Exceeding that value will result in a permanent failure. Second, the naive approach of iteratively assigning threshold voltages and testing the circuit is not only prohibitively time consuming, it will likely result in exceeding the maximum number of write cycles. This paper presents a novel solution to these two problems, ensuring that all manufactured circuits that fail in the manner described above can be corrected and made to function properly.

4.3 A Challenging Use Case

4.3.1 A Tunable Binary Perceptron

In this section we present a brief description of the circuit architecture of a binary neuron Wagle *et al.* (2019). Figure 4.1 shows the circuit, which is referred

to as *flash threshold logic* (FTL). It is a digital-analog-digital circuit that implements a special class of Boolean functions called threshold functions. A Boolean function $f(x_1, x_2, \cdots, x_n)$ is called a threshold function if there exist weights w_i for $i = 1, 2, \cdots, n$ and a threshold T[1] such that

$$f(x_1, x_2, \cdots x_n) = 1 \leftrightarrow \sum_{i=1}^{n} w_i x_i \geq T, \tag{4.1}$$

where $\sum$ denotes the arithmetic sum. Thus, a threshold function can be represented as $(W, T) = [w_1, w_2, \cdots, w_n; T]$. An example of a threshold function is $f(a, b, c, d, e) = ab \vee acd \vee bcd \vee ace \vee bce \vee ade \vee bde$, with $[w_1, w_2, w_3, w_4, x_5; T] = [2, 2, 1, 1, 1; 4]$. A 5-input FTL cell can implement all 117 threshold functions of 5 or fewer variables Muroga (1971).

The FTL circuit shown in Figure 4.1 consists of 4 main functional blocks – (1) a sense amplifier (SA), (2) an input network consisting of a left input network (LIN), a right input network (RIN), (3) an output latch (LA) and (4) the programming interface. The programming interface will not be described here as details are available in Wagle *et al.* (2019). The LIN and RIN each consist of n pairs of devices in parallel, each pair consisting of a flash transistor in series with an nFET. The current through each pair can be controlled by the (tunable) threshold voltage of the flash transistor in that pair. The threshold voltages of the flash transistors serve a proxy for the weights of a threshold function defined in Equation 4.1. Note that the weights and threshold in Equation 4.1 are distributed across both the LIN and RIN.

The inputs $(x_1, x_2, \cdots x_n)$ are mapped to the gate inputs of the LIN $(\ell_1, \ell_2, \cdots, \ell_n)$ and RIN $((r_1, r_2, \cdots, r_n)$. Let G_L and G_R denote the conductivity of the LIN and RIN, respectively. Their magnitudes depend on the values of the logic inputs, and the physical parameters of all the devices. For a given threshold function, the threshold voltages of the flash transistors $\widetilde{V_T} = (V_{\ell 0}, V_1, V_2, \cdots, V_n, V_{r0})$ in the LIN and RIN are

[1] W.L.O.G. the weights w_i and threshold T can be integers Muroga (1971).

set in such a way as to guarantee that $G_L > G_R$ for $f(x_1, x_2, \cdots x_n) = 1$, and $G_L < G_R$ for $f(x_1, x_2, \cdots x_n) = 0$. The operation of the circuit is as follows. When $CLK = 0$, the circuit is in reset and the outputs are $(N_1 = 1, N_2 = 1)$. This has no effect on the output latch. Now suppose inputs are applied to the nFETs in the LIN and RIN, such that $G_L > G_R$, and then $CLK : 0 \rightarrow 1$. Then the outputs $(N_1, N_2) = (0, 1)$. This will set the output latch, indicating that the function evaluated to a 1. If $G_L < G_R$, then $(N_1 = 1, N_2 = 0)$, and the output latch will be reset. Note that the mapping $[W, T] \rightarrow \widetilde{V}_T$ is one-to-many. That is, different sets of threshold voltages of the flash devices can implement the same threshold function. Moreover, the same function implemented on two different FTLs on a chip might have to be programmed with different threshold voltages depending on parasitics associated with each FTL cell.

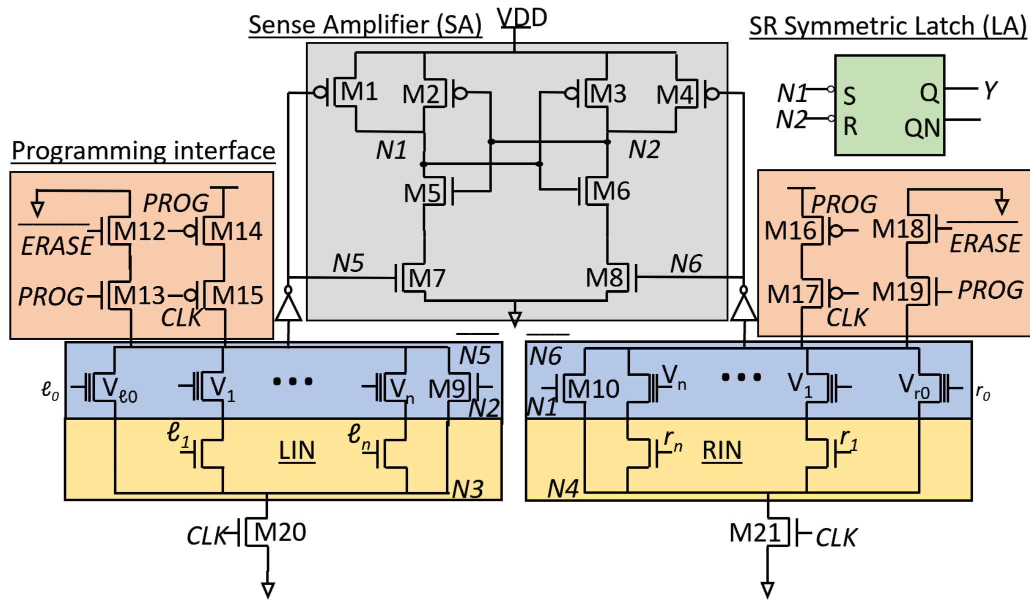


Figure 4.1: The Architecture of the FTL Cell with Four Main Components: The Left Input Network (LIN), the Right Input Network (RIN), a Sense Amplifier (SA) and an Output Latch (LA). The LIN and RIN Consist of Two Sets of Inputs $(\ell_1, \cdots, \ell_n)$ and $(r_1, \cdots, r_n)$, Respectively, With Each Input in Series with a Flash Transistor.

4.3.2 Sources of Variations in the FTL Circuit

In an ideal circuit, if $G_L > G_R$, the SA evaluates to 1, otherwise it evaluates to 0. Manufacturing variations can disrupt this relation. The condition $G_L = G_R$ is considered as an erroneous metastable state as shown in Figure 4.2. The robustness of the implemented function is determined based on how close the G_L and G_R are to the metastability line. Even if the G_L and G_R stay the same in the presence of process variations, the variations in the SA can change the metastability line. The FTL cell can be made to be robust to process, voltage and temperature variations by tuning the threshold voltages of the flash transistors to compensate for the variations in the other devices.

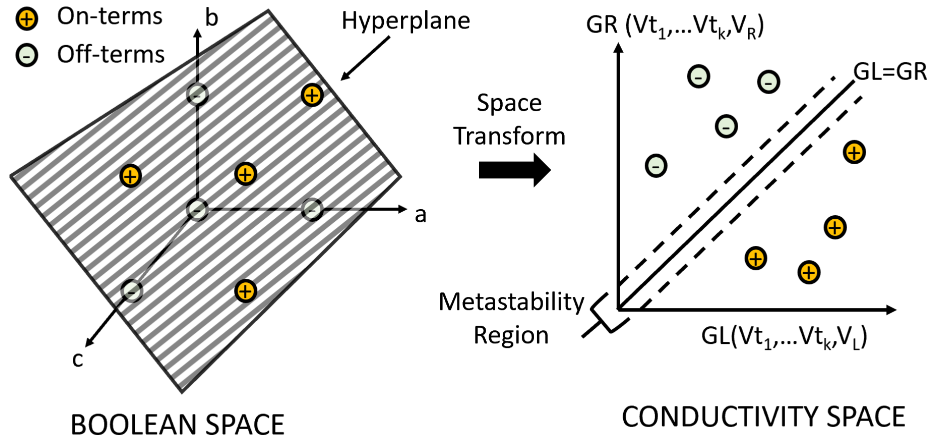


Figure 4.2: Transformation from Boolean Space to Conductivity Space; Hyperplane Gets Converted into a Line.

4.3.3 Programming the FTL Circuit

In this section, we describe how the threshold voltages of the flash transistors are determined to realize a given threshold function. That is, how the mapping $[W, T] \rightarrow \widetilde{V}_T$ is performed.

Given the truth table (TT) of a target function $f[W, T]$, the modified Perceptron Learning Algorithm (PLA), based on Rosenblatt (1958), is employed to determine the set of Vt(s) needed to implement f on an FTL. The operation of PLA is as follows: First, all minterms are applied as inputs to the FTL, and the response of the FTL is recorded. This response is compared to the value in TT. If there is a mismatch in the response for even a single minterm (m_i), then for all the ON inputs in that minterm, the threshold voltages of the corresponding transistors are adjusted by δ using Equation 4.2.

$$V_i^{k+1} = \begin{cases} V_i^k - \delta m_i & m_i \cdot W \geq T \\ V_i^k + \delta m_i & m_i \cdot W < T. \end{cases} \tag{4.2}$$

After adjusting $\widetilde{V}_T$, the response of the FTL for each minterm is recorded again. This process continues until the output response matches the TT. In cases where it is not possible to satisfy the TT using the V_i alone, the PLA will resort to adjusting $V_{\ell 0}$ and V_{r0} using the same principle as in Equation 4.2. $V_{\ell 0}$ and V_{r0} are the transistors corresponding to the threshold value T in Equation 4.1. The PLA can be used to find the $\widetilde{V}_T$ for any FTL circuit with process variations as they can be adjusted to counter the imbalance in the evaluation of the SA. The result of the PLA is a set of threshold voltages denoted by Vt^{set}. This will be referred to as a nominal Vt^{set}.

4.4 Post Fabrication Weight Tuning

4.4.1 The Issues of On-chip PLA

An ideal FTL cell can be programmed using the nominal Vt^{set} to implement a target function. Consider several manufactured FTL circuits. At first, the nominal Vt^{set} is applied to all these circuits. The output of some circuits match the intended

TT and are considered as passing instances. However, the remaining instances fail due to process variations. To achieve a 100% yield, the weights of the failed instances should be re-tuned and a new Vt^{set} needs to be generated for every instance using PLA to counter the variations. From the device perspective, the flash transistors in the FTL cells have limited write and erase cycles. Using the PLA on-chip to determine the Vt^{set} would require wiping out the stored charges on the flash transistors several times. This technique is impractical and should therefore be avoided.

It is necessary to determine a working Vt^{set} for an FTL without exceeding the maximum permissible write-erase cycles of the flash. This can be performed by characterizing several failing instances in the presence of process variations in simulation. A new Vt^{set} can then be generated to counter the effect of variations, and applied directly on the fabricated chip.

4.4.2 Alternate Solution to On-chip PLA

The most efficient way to program the FTL cells on a chip is to identify the cause of failing instances in one write cycle and fix them using a working Vt^{set} in the next write cycle. To use this approach, a database needs to be constructed with two entries: the first entry would contain the types of failures and the second entry contains the Vt^{sets} to fix those failures. Such a database needs to be generated using a statistical methodology, such that it assures a 100% yield upon fabrication. Several instances of the FTL cell need to be characterized, to thoroughly cover the potential failures and the corresponding Vt^{set} to fix them. This database is called $FabDB$ (post-Fabrication DataBase).

The construction of $FabDB$ starts with generating the nominal Vt^{Set} using PLA for an FTL cell, assuming the nominal or average values of all the variational parameters. For example, the Vt^{Set} of a 5-input FTL cell consists of 6 Vt values:

$[Vt^a, Vt^b, Vt^c, Vt^d, Vt^e; Vt^T]$ that need to be programmed. Although most of the FTL cells work with the nominal Vt^{Set}, some still fail due to process variations. A failure is identified by observing the minterm(s), whose output does not match with the output of the target threshold function. Specifically, a failure type is a unique subset of minterms in a TT that fail. By observing the failure types, one can determine whether the effective weights of the transistors had increased or decreased due to process variations, and consequently the cause of failures. For example, consider a 5-input threshold function $4a + b + c + d + e >= 5$, to be implemented on an FTL cell. Assume that due to process variations, the effective weight of input a increases to 1 after programming. The FTL cell would implement $5a+b+c+d+e >= 5$. The minterm $[a, b, c, d, e] = [10000]$ fails but the remaining minterms pass. The failure type here is $\{10000\}$. Since $G_L > G_R$ for minterm [10000], PLA decreases the weight of a by increasing the Vt of the flash transistor connected to a by δ volts, thereby making $G_L < G_R$. Thus, we have to first identify all the potential failure types before fabrication and the corresponding working Vt^{Set}. These will be stored in a database called $FabDB$. Table 4.1 shows a sample $FabDB$ for a 5-input FTL. Here, the Vt^{Sets} are generated by running PLA to fix different failure types. In Table 4.1, the failures occur only due to the variations at input a.

After generating $FabDB$, the steps to program a fabricated FTL chip is as follows:

1. The FTL cells of the fabricated chip are programmed with the nominal Vt^{Set}.
2. The ones that fail, their failure types are extracted according to the target TT.
3. A new Vt^{Set} corresponding to each failure type is retrieved from $FabDB$ and used to program the failed instances again. This new Vt^{Set} makes the failed instances to function correctly.

Table 4.1: An Example of $FabDB$ That Consists of the Failure-types and Their Respective Vt^{Set}. The Order of the Minterms Is [a,b,c,d,e]. For Example, the Failure Type 10000 Corresponds to the Threshold Function a Instead of $ab + ac + ad + ae$.

Unique Failure Type	New Vt Solution $[Vt^a, Vt^b, Vt^c, Vt^d, Vt^e; Vt^T]$
{10000, 00011 , ... , 01111}	{0.3, 0.5, 0.5, 0.5, 0.5; 0.4}
{10000}	{0.4, 0.5, 0.5, 0.5, 0.5; 0.4}
{} (No failure)	{0.5, 0.5, 0.5, 0.5, 0.5; 0.4}

4.4.3 Database Construction for FTL Programming

This section proposes an iterative procedure to construct $FabDB$. This is to determine a group of Vt^{Sets} needed to program all the potential FTL instances to achieve 100% yield for a target function. Figure 4.3 depicts the structure of this procedure that consists of two consecutive flows.

Flow-1: The flow shown in Figure 4.3 (left) starts with the nominal Vt^{Set} and several FTL instances generated by varying the process parameters. It simulates the circuits to store all the potential failure types that will be observed when a fabricated chip is programmed with the nominal Vt^{Set}. The circuit instances and their corresponding failure types are stored in the database $CktFT$ (Circuit-to-Failure-Type). Table 4.2 shows an example of $CktFT$ for three FTL instances. The unique failure types are then extracted and stored in the first entry of $FabDB$. Additional information needed to fix the failure types in $FabDB$ is added to the database in Flow-2.

Flow-2: The flow shown in Figure 4.3 (right) determines the working Vt^{Sets} for fixing the unique failure-types extracted in Flow-1. The flow starts with the same nominal Vt^{Set} and circuit instances used in Flow-1. The circuits are simulated to determine

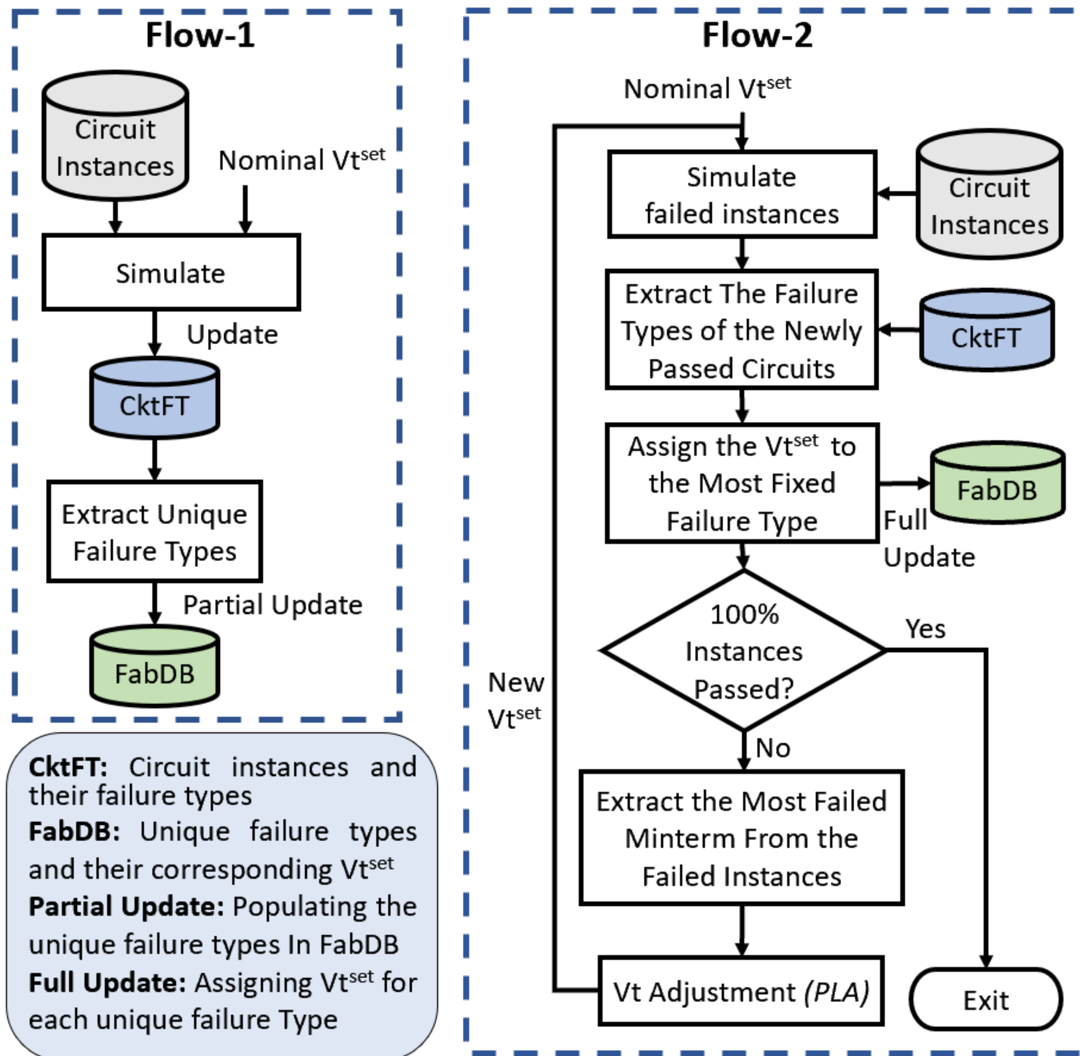


Figure 4.3: Proposed Procedure for *FabDB* Construction. Flow-1 Simulates the Instances Programmed with the Nominal Vt^{Set} and Generates the Failure Types. This is Stored in *CktFT*. The Unique Failure Types are Stored in the First Entry of *FabDB*. Flow-2 Uses a Combination of Circuit Simulation and PLA to Find a Working Vt^{Sets} for the Unique Failure Types Which Is Stored the Second Entry of *FabDB*.

Table 4.2: An Example of *CktFT* Database for Three Instances of a 5-input FTL and Their Corresponding Failure Types.

FTL Instances	Failure Types
Instance-1	{10000, 00011 , ... , 01111}
Instance-2	{10000}
Instance-3	{} (No Failure)

whether their output responses match the intended TT. If a Vt^{Set} makes some/all of the failed instances to function correctly, it is stored in the second entry of $FabDB$ as a solution to the unique failure type that is fixed the most. To find the working Vt^{Sets} for the remaining failed instances, the most failed minterm is extracted from the output response. PLA makes appropriate modifications (see Section 4.3.3) and generates a new Vt^{Set}. This procedure is repeated until all the failed instances pass and all the unique failure types have a working Vt^{Set}. At the end this flow, $FabDB$ can be used to program the FTL cells in a fabricated chip and achieve a 100% yield. This is performed by observing the failure type of a fabricated cell and program it by applying a corresponding working Vt^{Set} from $FabDB$.

This procedure is significantly useful for fine-tuning the weights after fabrication. However, using transient HSPICE for the simulations in both the flows is compute-intensive and impractical as it needs to be repeated thousands of times. Next section proposes a significantly smaller and faster simulator to replace HSPICE.

4.5 The Proposed Stochastic Simulator

This section describes an stochastic simulator based on the aPC method Oladyshkin and Nowak (2012) discussed in Section 3.4. Replacing HSPICE with the

stochastic simulator not only makes the construction of $FabDB$ feasible but also expedites the procedure. The variations of the process parameters are first extracted from the TSMC model. One circuit instance is equivalent to one set of variations applied on the average process parameters. The stochastic simulator receives the variations in these parameters and produces the output voltage of node Q in the FTL cell.

As a concrete example, consider a transistor with two process parameter, ξ_1 and ξ_2 that represent L and W. The coefficients of the $0th$ and $1st$ order polynomials for each variable is expressed in Equation 4.3. The second order uni-variate polynomials whose coefficients are functions of the moments (μ) are expressed in Equation 4.4.

$$p_0^{(2)} = -1,\ p_1^{(2)} = -\mu_3,\ p_2^{(2)} = 1 \tag{4.3}$$

$$P^{(0)}(\xi) = 1,\ P^{(1)}(\xi) = \xi,\ P^{(2)}(\xi) = \xi^2 - \mu_3\xi - 1 \tag{4.4}$$

The multivariate polynomial basis for a $2nd$ order model are obtained by taking the cross product of the univariate polynomials as shown in Equation 4.5. There are six basis polynomials and six unknown coefficients (c_i), which are solved using least squares.

$$\Psi(\xi_1, \xi_2) = \{1,\ \xi_1,\ \xi_2,\ \xi_1\xi_2, \xi_1^2 - \mu_3^{(L)}\xi_1 - 1,\ \xi_2^2 - \mu_3^{(W)}\xi_2 - 1\} \tag{4.5}$$

$$V_Q(\boldsymbol{\xi}) = \sum_{i=1}^{\binom{N+d}{d}} c_i \Psi_i(\boldsymbol{\xi}). \tag{4.6}$$

In the FTL cell, $\boldsymbol{\xi} = [Vt_1, W_1, L_1, ..., Vt_n, W_n, L_n]$ for n transistors. The voltage of node Q is modeled using Equation 4.6, where N is the total number of parameters

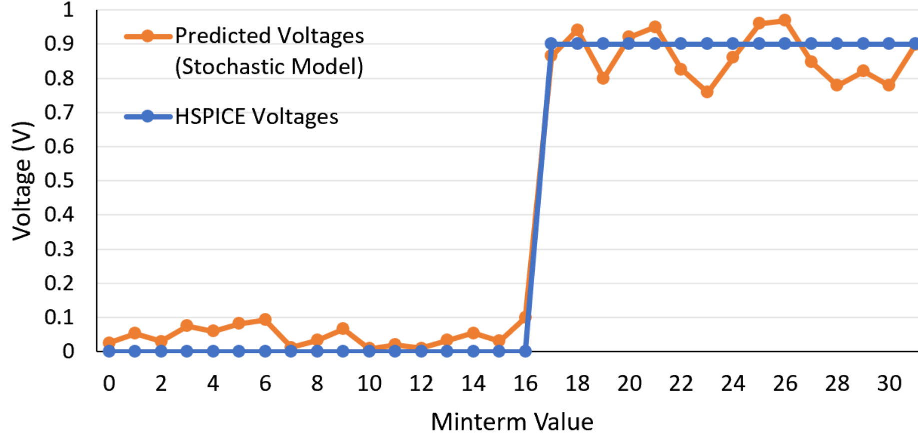


Figure 4.4: Output Voltage of a 5-input FTL Instance Simulated with the Stochastic Simulator and HSPICE. The Target Function Is [4,1,1,1,1;5]. The Voltages Are Then Thresholded to 0 and 1 to Generate The TT.

(i.e. $n \times 3$) and d is the order of the polynomial expansion. The predicted voltages are then thresholded to generate the corresponding TT of the target function.

Figure 4.4 illustrates an example of an FTL instance simulated by both the stochastic simulator and HSPICE to generate V_Q. The following observations further simplify and enhance the accuracy of the simulator.

Observation 1: During evaluation phase, the SA of an FTL compares the parameters G_L and G_R. Hence, the functional correctness of an FTL cell depends only on these parameters and the position of the metastability line in the SA, and not on the latch transistors. When FTL finishes its evaluation, voltage at $N2$ and Q are equal. Hence, the parameters of the latch transistors can be omitted to simplify the simulator.

Observation 2: The Tox variations (δ_{Tox}) in the TSMC model were three orders of magnitude smaller than that of the other parameters. It was observed that δ_{Tox} was adding noise to the constructed model. Thus, the prediction accuracy was further improved by 8% by omitting Tox from $\boldsymbol{\xi}$.

Observation 3: The nominal Vt^{Set} is intended to implement a single threshold func-

tion on an FTL cell with average process variations. However, the process variations can cause the FTL instances to implement a varying set of threshold functions. It was observed that the typical size of the dominant functions in this function set was four. In the example shown in Figure 4.5, the nominal Vt^{Set} for the target function [4,1,1,1,1;5] produced a distribution of threshold functions over several FTL instances. Four functions including the target dominate the set.

Based on the above observations, the variations in L, W and Vt of the remaining 30 transistors are modeled in the stochastic simulator leading to a total of 90 random variables. Based on empirical evaluation, the 2nd order model coefficients produce the best prediction accuracy. To compute the coefficients (c_i) using least squares, the number of samples need to be slightly more than the number of coefficients. Thus, 8000 samples were used for model construction for $\binom{92}{2}$ coefficients. Computation of the c_i is a one time cost and takes negligible time. Constructing the model by taking the majority of the samples from the dominant functions improves the accuracy as it removes the noise from the model. The evaluation of the simulator on the remaining circuits which were not part of the model construction showed an extra 10% improvement in the accuracy.

4.6 Experimental Results

4.6.1 Experimental Setup

In the conventional PLA, HSPICE is used to simulate the FTL cell repeatedly to find a working Vt^{set}. Although HSPICE predicts the output response accurately, it requires a large amount of memory and is compute-intensive. Moreover, for $FabDB$ construction, using HSPICE is not feasible due to thousands of simulations. The proposed stochastic simulator is capable of reducing the dependency on HSPICE without

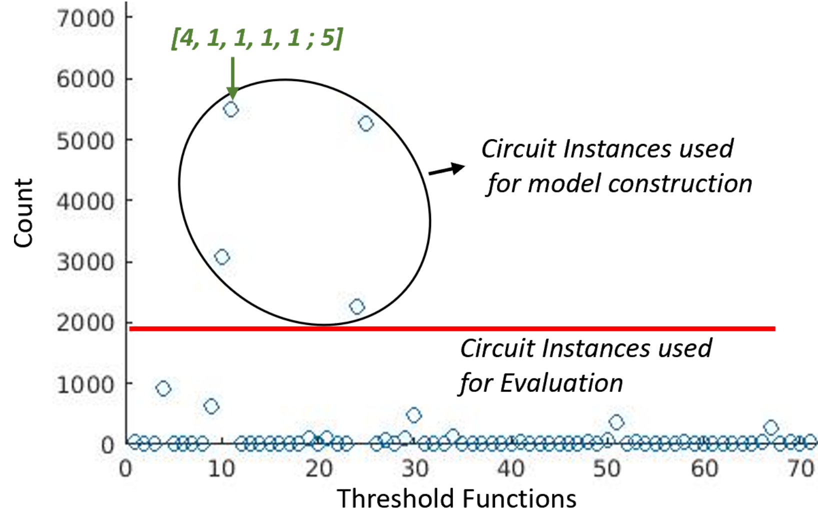


Figure 4.5: Example Distribution of Threshold Functions When the Instances Were Programmed with the Nominal Vt^{Set} For the Target Function [4,1,1,1,1;5]. The Four Most Frequent Functions Are Used for Model Construction. The Model Is Evaluated on the Remaining Circuit Instances. This Strategy Boosts the Accuracy by an Extra 10%.

loss of accuracy, and is significantly faster. To demonstrate these advantages, PLA-based experiments were set up using TSMC 40nm LP Technology. The PVT corner setting was $[P, V, T] = [TT, 0.9V, 25°C]$. The 5-input threshold function [4,1,1,1,1;5] was set as the target function. It contains inputs of equal and unequal weights and implements a sophisticated threshold function. These experiments can be extended to implement other threshold functions as well.

4.6.2 Results

1) Stochastic Simulator vs. HSPICE: In this experiment, PLA is used to find the working Vt^{Set} for 100 FTL cell instances. The stochastic simulator is used for rapid execution of the PLA iterations. It then uses HSPICE in its final iterations before converging to the working Vt^{Set}. This is to maintain the reliability and accuracy of the PLA. The speed up is computed based on the number of iterations when

HSPICE is used without the stochastic simulator, over the number of iterations required with the stochastic simulator. Note that the reported speed-up is based on the required number of HSPICE iterations and is machine-independent. Figure 4.6 shows the speed up achieved by employing the stochastic simulator and saving HSPICE iterations. This experiment is further summarized in Table 4.3.

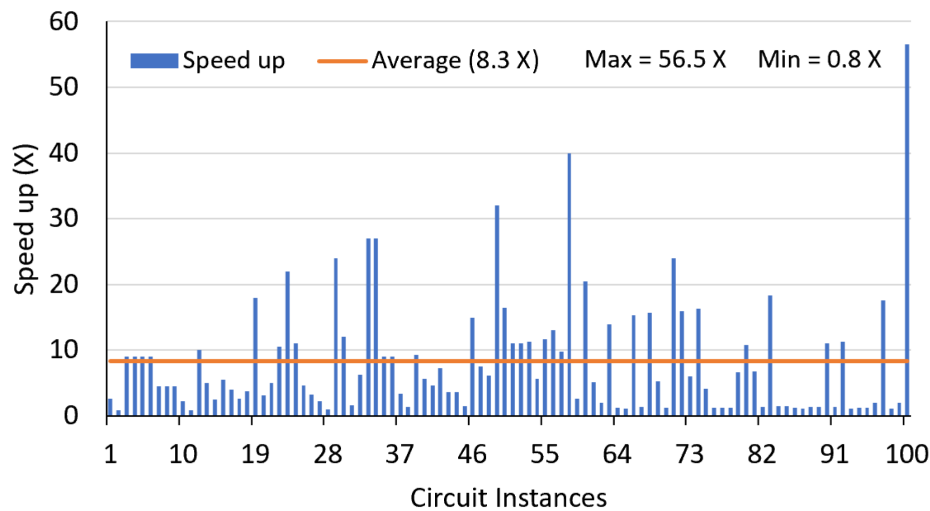


Figure 4.6: The Speed up Achieved by Employing the Stochastic Simulator to Reduce the HSPICE Iterations in PLA. Average Speedup of 8.3x Is Achieved with the Maximum Being 56.5X.

Table 4.3: The Number of Saved HSPICE Iterations and the Speed up Achieved by Employing the Stochastic Simulator. This Experiment Is Performed on 100 Circuit Instances to Find Their Working Vt^{sets}.

	Max	Min	Average
Saved HSPICE Iterations	222	-9	24.5
Speed-up (X)	56.5	0.8	8.3

To compare the execution time of the stochastic simulator against HSPICE, sets of parallel FTL simulations were run on the same machine. Figure 4.7 shows more than three orders of magnitude improvement when parallel simulations are performed using the stochastic simulator. Due to the memory limitations of the machine, simulating more than 50 parallel circuits in HSPICE was not possible. Additional HSPICE runs would need to be grouped and scheduled sequentially. However, the stochastic simulator was able to handle several thousands of parallel runs in negligible time.

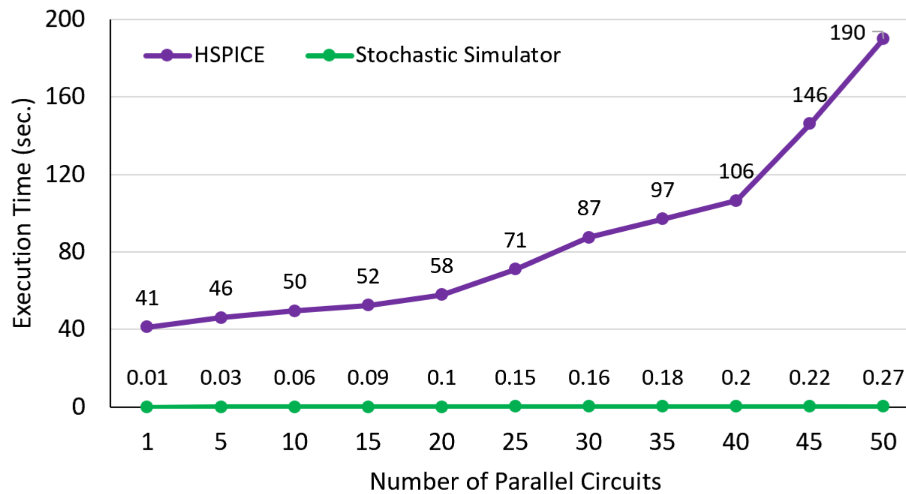


Figure 4.7: The Execution Time (Seconds) of Parallel Circuits Simulated in Both HSPICE and the Stochastic Simulator on the Same Machine.

2) Stochastic Simulator vs. HSPICE for constructing FabDB: An experiment is conducted to construct $FabDB$ for 1000 circuits using Flow-1 and Flow-2. This was not possible using HSPICE alone. However, with the stochastic simulator's speed, simplicity and negligible memory requirements, several iterations of 1000-circuit simulations are feasible even on a low-end machine. At first, in Flow-1, all the circuits were simulated using the stochastic simulator to update $CktFT$ and the first column of $FabDB$. The number of unique failure types were found to be 104. Then, the procedure in Flow-2 was followed to simulate the failed instances,

find a working Vt^{set} for each of them, and update the second column of $FabDB$, accordingly. To maintain the accuracy, the Vt^{sets} in $FabDB$ is further fine-tuned with HSPICE. This is done by extracting an FTL instance and its Vt_{Set} from $CktFT$ and $FabDB$, respectively. Using these as the starting point, an HSPICE-based PLA is used to generate a new fine-tuned Vt_{Set}. This new Vt_{Set} is updated in the second column of $FabDB$ as a solution for that failure type. Figure 4.8 illustrates the speed up achieved constructing the $FabDB$ with no loss of accuracy.

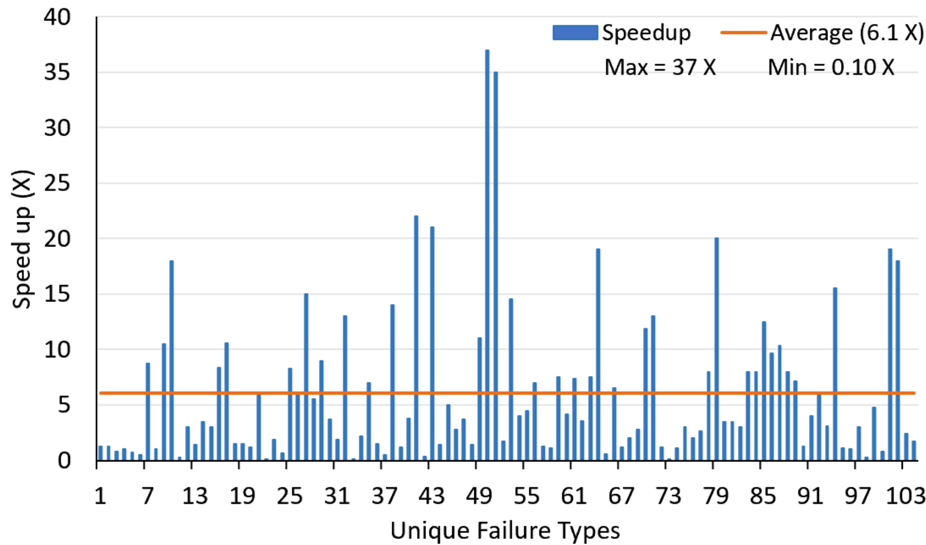


Figure 4.8: The Speed up Achieved by Employing the Stochastic Simulator to Reduce the HSPICE Iterations For $FabDB$ Construction. Average Speedup of 6.1x Is Achieved with the Maximum Being 37x. Constructing $FabDB$ Using Only HSPICE Is Impractical Due to the Several Simulations Required to Generate the Failure Types.

4.7 Chapter Summary

This paper presents a novel statistical methodology for programming a flash-based binary perceptron (called FTL cell) after fabrication. In the presence of process variations, to achieve a 100% yield, the weights of the failed cells can be tuned using

the perceptron learning algorithm. Since this is impractical due to limited number of write and erase cycles in flash, the proposed methodology generates a database to counter the effect of process variations. The procedure to generate such a database requires thousands of FTL simulations, which is extremely compute intensive. A stochastic simulator was designed based on an extension to polynomial chaos expansion to handle such an extreme computation load, which is otherwise infeasible using HSPICE. The simulator is an extension of the Polynomial Chaos Expansion which models the intra-die variations of an FTL. Due to the high number of device parameters and dimensionality of the problem, thorough exploration of the parameters and empirical experiments were conducted to extract the important parameters without sacrificing the accuracy of the simulator. The results demonstrate that the stochastic simulator can reduce the dependency on the HSPICE for weight tuning without loss of accuracy while achieving a speedup of 56.5X when compared with HSPICE.

Chapter 5

CONCLUSION

To evaluate the design objectives and constraints of many engineering problems, extensive simulations need to be performed. More often than not, due to the complexity of the system model ($\mathcal{M}$), these simulations can be expensive in terms of computation time or even infeasible to perform. One approach to overcome this issue is to construct a *surrogate model* ($\widetilde{\mathcal{M}}$), which is an approximation of the original model ($\mathcal{M}$) that mimics the behavior of $\mathcal{M}$. The focus of this work is on the *data-driven* or *black-box* surrogate models, in which empirical approximations of the output are performed given the input parameters. Some of the examples are Gaussian processes, neural networks and polynomial chaos expansions. In the broad field of machine learning (ML), recently artificial neural networks (NN) have re-emerged as a popular method for constructing data-driven surrogate models. Although NN models have achieved excellent accuracy and are used in a wide variety of applications, they have their own challenges. In this work, we address two common challenges, namely, the need for: (1) *hardware acceleration* and (2) *uncertainty quantification (UQ) in the presence of input variability.*

The high demand in the inference phase of deep NN models in both the cloud servers and edge devices calls for the design of low power custom hardware accelerators. The first part of this work describes the design of an e̲nergy-efficient **L̲S̲**TM **a̲**ccelerator, referred to as ELSA. The overarching goal of this work is to aggressively reduce the power consumption and area of the LSTM components using approximate computing, and then use architectural level techniques to boost the performance. First, we design and employ low power and compact computation units for the LSTM.

Some of these modules use approximate calculations, which consume much less power than conventional implementations but incur a high execution time penalty. Second, to recover the throughput loss and achieve higher energy efficiency, we develop efficient scheduling techniques that include overlapping of the computations at multiple levels – from the lowest level modules up to the application level. Our proposed design was prototyped on a Xilinx FPGA and then synthesized and placed and routed in 65nm CMOS technology as an ASIC. The results demonstrate that ELSA is 1.2X and 3.6X higher energy-efficient and area-efficient, respectively than the baseline LSTM.

Although deep NN models are increasingly being deployed in safety-critical applications, the reported successes hide a severe threat–the lack of an accurate measure of uncertainty associated with the prediction. Quantifying the uncertainty in a prediction has several practical uses: handing over the control to a human or transitioning to a safe mode by an autonomous vehicle or a robot, suggesting further analysis in medical diagnosis, etc. The second part of this work aims to develop a robust framework based on an alternate data-driven surrogate model referred to as *polynomial chaos expansion* (PCE) for addressing uncertainty quantification (UQ). In contrast to many existing approaches, no assumptions (e.g., Gaussian processes or prior distributions on the parameters, etc) are made on the elements of the function space and the UQ is a function of the expansion coefficients. In addition, the sensitivity of the model output with respect to any subset of the input variables can be computed analytically by post-processing the PCE coefficients. This provides a systematic and incremental method to pruning or changing the order of the model. The proposed method for UQ is applied on several real-world applications from different domains and is extended for classification tasks as well.

Albericio, J., A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov and A. Moshovos, "Bit-pragmatic deep neural network computing", in "Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture", MICRO-50 '17, pp. 382–394 (ACM, New York, NY, USA, 2017), URL http://doi.acm.org/10.1145/3123939.3123982.

Amodei, D. and D. Hernandez, "Openai", URL https://openai.com/blog/ai-and-compute/ (2018).

Anwar, S., K. Hwang and W. Sung, "Structured pruning of deep convolutional neural networks", J. Emerg. Technol. Comput. Syst. **13**, 3, URL https://doi.org/10.1145/3005348 (2017).

Archer, G. E. B., A. Saltelli and I. M. Sobol, "Sensitivity measures,anova-like techniques and the use of bootstrap", Journal of Statistical Computation and Simulation **58**, 2, 99–120, URL https://doi.org/10.1080/00949659708811825 (1997).

Asher, M. J., B. F. W. Croke, A. J. Jakeman and L. J. M. Peeters, "A review of surrogate models and their application to groundwater modeling", Water Resources Research **51**, 8, 5957–5973, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2015WR016967 (2015).

Augustin, F., A.Gilg, M. Paffrath, P. Rentrop and U. Wever, "Polynomial chaos for the approximation of uncertainties: Chances and limits", European Journal of Applied Mathematics **19**, 149–190 (2008).

Azari, E. and S. Vrudhula, "An energy-efficient reconfigurable lstm accelerator for natural language processing", in "2019 IEEE International Conference on Big Data (Big Data)", pp. 4450–4459 (2019).

Azari, E. and S. Vrudhula, "Elsa: A throughput-optimized design of an lstm accelerator for energy-constrained devices", ACM Trans. Embed. Comput. Syst. **19**, 1, URL https://doi.org/10.1145/3366634 (2020).

Babuška, I., F. Nobile and R. Tempone, "A stochastic collocation method for elliptic partial differential equations with random input data", SIAM Rev. **52**, 2, 317–355, URL http://dx.doi.org/10.1137/100786356 (2010).

Beluch, W. H., T. Genewein, A. Nurnberger and J. M. Kohler, "The power of ensembles for active learning in image classification", in "2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 9368–9377 (2018).

Betz, V. and J. Rose, "VPR: a new packing, placement and routing tool for fpga research", in "Field-Programmable Logic and Applications", edited by W. Luk, P. Y. K. Cheung and M. Glesner, pp. 213–222 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1997).

Bhardwaj, S., S. Vrudhula and A. Goel, “A unified approach for chip statistical timing and leakage analysis of nanoscale circuits considering intradie process variations”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **27**, 10, 1812–1825 (2008).

Blundell, C., J. Cornebise, K. Kavukcuoglu and D. Wierstra, “Weight uncertainty in neural networks”, in “Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37”, ICML’15, p. 16131622 (JMLR.org, 2015).

Brynjolfsson, E. and A. Mcafee, “The business of articifial intelligence”, URL `http://mlr.cs.umass.edu/ml/` (2017).

Bucila, C., R. Caruana and A. Niculescu-Mizil, “Model compression”, in “Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining”, KDD 06, p. 535541 (Association for Computing Machinery, New York, NY, USA, 2006), URL `https://doi.org/10.1145/1150402.1150464`.

Cameron, R. H. and W. T. Martin, “The orthogonal development of non-linear functionals in series of fourier-hermite functionals”, Annals of Mathematics **48**, 2, 385–392, URL `http://www.jstor.org/stable/1969178` (1947).

Cao, S., C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu and L. Zhang, “Efficient and effective sparse lstm on fpga with bank-balanced sparsity”, in “Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays”, FPGA ’19, pp. 63–72 (ACM, New York, NY, USA, 2019), URL `http://doi.acm.org/10.1145/3289602.3293898`.

Chang, A. X. M. and E. Culurciello, “Hardware accelerators for recurrent neural networks on fpga”, in “2017 IEEE International Symposium on Circuits and Systems (ISCAS)”, pp. 1–4 (2017).

Chang, A. X. M., B. Martini and E. Culurciello, “Recurrent neural networks hardware implementation on FPGA”, CoRR URL `http://arxiv.org/abs/1511.05552` (2015).

ChangWu Huang, B. R., Abdelkhalak El Hami, “Overview of structural reliability analysis methods part i: Local reliability methods”, Uncertainties and Reliability of Multiphysical Systems **1**, Optimization and Reliability (2017).

Cho, K., B. van Merriënboer, D. Bahdanau and Y. Bengio, “On the properties of neural machine translation: Encoder–decoder approaches”, in “Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation”, pp. 103–111 (Association for Computational Linguistics, Doha, Qatar, 2014), URL `https://www.aclweb.org/anthology/W14-4012`.

Choukroun, Y., E. Kravchik, F. Yang and P. Kisilev, “Low-bit quantization of neural networks for efficient inference”, (2019).

Conti, F., L. Cavigelli, G. Paulin, I. Susmelj and L. Benini, “Chipmunk: A systolically scalable 0.9 mm2, 3.08gop/s/mw @ 1.2 mw accelerator for near-sensor recurrent neural network inference”, in “2018 IEEE Custom Integrated Circuits Conference (CICC)”, pp. 1–4 (2018).

Courbariaux, M. and Y. Bengio, “Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1”, CoRR **abs/1602.02830**, URL `http://arxiv.org/abs/1602.02830` (2016).

Damianou, A. C. and N. D. Lawrence, “Deep gaussian processes”, (2012).

Dario Amodei et al., “Deep speech 2: End-to-end speech recognition in english and mandarin”, in “Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48”, ICML’16, pp. 173–182 (JMLR.org, 2016), URL `http://dl.acm.org/citation.cfm?id=3045390.3045410`.

Das, S. and S. Han, “Neuraltalk on embedded system and GPU-accelerated RNN”, (2015).

Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, (2018).

Eldred, M. and J. Burkardt, “Comparison of non-intrusive polynomial chaos and stochastic collocation methods for uncertainty quantification”, in “47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition”, (2012), URL `https://arc.aiaa.org/doi/abs/10.2514/6.2009-976`.

Ernst, O. G., A. Mugler, H.-J. Starkloff and E. Ullmann, “On the convergence of generalized polynomial chaos expansions”, ESAIM: Mathematical Modelling and Numerical Analysis **46**, 2, 317339 (2012).

Ferreira, J. C. and J. Fonseca, “An fpga implementation of a long short-term memory neural network”, in “2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)”, pp. 1–8 (2016).

Fowers, J., K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung and D. Burger, “A configurable cloud-scale dnn processor for real-time ai”, in “Proceedings of the 45th Annual International Symposium on Computer Architecture”, ISCA ’18, pp. 1–14 (IEEE Press, Piscataway, NJ, USA, 2018), URL `https://doi.org/10.1109/ISCA.2018.00012`.

Gal, Y. and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”, in “Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48”, ICML16, p. 10501059 (JMLR.org, 2016).

Ghahramani, Z., “Probabilistic machine learning and artificial intelligence”, Nature **521**, 7553, 452–459, URL `https://www.ncbi.nlm.nih.gov/pubmed/26017444/`, on Probabilistic models (2015).

Ghanem, R. and P. Spanos, *Stochastic Finite Elements: A Spectral Approach* (1991).

Ghanta, P. and S. Vrudhula, "Analysis of power supply noise in the presence of process variations", IEEE Design & Test of Computers **24**, 3, 256–266 (2007).

Goel, A., S. Vrudhula, F. Taraporevala and P. Ghanta, "Statistical timing models for large macro cells and IP blocks considering process variations", IEEE Transactions on Semiconductor Manufacturing **22**, 1, 3–11 (2009).

Graves, A., "Practical variational inference for neural networks", in "Advances in Neural Information Processing Systems", edited by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira and K. Q. Weinberger, vol. 24 (Curran Associates, Inc., 2011), URL `proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf`.

Graves, A., *Supervised sequence labelling with recurrent neural networks*, vol. 385 (Springer, 2012).

Graves, A. and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures", Neural Networks **18**, 602–610 (2005).

Guan, Y., H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates", in "2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)", pp. 152–159 (2017a).

Guan, Y., Z. Yuan, G. Sun and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks", in "2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)", pp. 629–634 (2017b).

Han, S., J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang and W. B. J. Dally, "Ese: Efficient speech recognition engine with sparse lstm on fpga", in "Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", FPGA '17, pp. 75–84 (ACM, New York, NY, USA, 2017), URL `http://doi.acm.org/10.1145/3020078.3021745`.

Han, S., H. Mao and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding", in "4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings", edited by Y. Bengio and Y. LeCun (2016), URL `http://arxiv.org/abs/1510.00149`.

Han, S., J. Pool, J. Tran and W. J. Dally, "Learning both weights and connections for efficient neural networks", in "Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1", NIPS'15 (MIT Press, Cambridge, MA, USA, 2015).

HASTINGS, W. K., "Monte Carlo sampling methods using Markov chains and their applications", Biometrika **57**, 1, 97–109 (1970).

Hazelwood, K., S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective", in "2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)", pp. 620–629 (2018).

He, K., X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition", in "2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", pp. 770–778 (2016).

Hernández-Lobato, J. M. and R. P. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks", in "Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37", ICML'15, p. 18611869 (JMLR.org, 2015).

Hinton, G., O. Vinyals and J. Dean, "Distilling the knowledge in a neural network", in "NIPS Deep Learning and Representation Learning Workshop", (2015), URL `http://arxiv.org/abs/1503.02531`.

Hinton, G. E. and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights", in "Proceedings of the Sixth Annual Conference on Computational Learning Theory", COLT '93, p. 513 (Association for Computing Machinery, New York, NY, USA, 1993), URL `https://doi.org/10.1145/168304.168306`.

Hochreiter, S., Y. Bengio, P. Frasconi and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies", in "A Field Guide to Dynamical Recurrent Neural Networks", (2001).

Hochreiter, S. and J. Schmidhuber, "Long short-term memory", Neural Comput. **9**, 8, 1735–1780, URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735` (1997).

Hu, X., R. G. Harber and S. C. Bass, "Expanding the range of convergence of the cordic algorithm", IEEE Trans. Comput. **40**, 1, 13–21, URL `https://doi.org/10.1109/12.67316` (1991).

Hwang, K. and W. Sung, "Single stream parallelization of generalized lstm-like rnns on a gpu", in "2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)", pp. 1047–1051 (2015).

Ince, M., S. Ozev and S. Vrudhula, "Statistical Library Characterization Using Arbitrary Polynomial Chaos", in "Proc. IEEE Latin American Symp. on Circuits and Systems (LASCAS)", (Bariloche, Argentina, 2017).

Jaynes, E. T., *Probability Theory: The Logic of Science* (Cambridge University Press, 2003).

Judd, P., J. Albericio, T. Hetherington, T. M. Aamodt and A. Moshovos, "Stripes: Bit-serial deep neural network computing", in "2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)", pp. 1–12 (2016).

K. Greff et. al., "LSTM: A search space odyssey", IEEE Trans. on Neural Networks and Learning Systems , 99, 1–11 (2017).

Karpathy, A. URL https://github.com/karpathy/char-rnn (2016).

Karpathy, A., G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-scale video classification with convolutional neural networks", in "2014 IEEE Conference on Computer Vision and Pattern Recognition", pp. 1725–1732 (2014).

Kaya, H. and P. Tufekci, "Local and global learning methods for predicting power of a combined gas and steam turbine", in "Proceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering", pp. 13 – 18 (2012).

Kingma, D. P. and M. Welling, "Auto-Encoding Variational Bayes", in "2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings", (2014).

Krizhevsky, A., "Learning multiple layers of features from tiny images", Tech. rep. (2009).

Krizhevsky, A., I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in "Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1", NIPS12, p. 10971105 (Curran Associates Inc., Red Hook, NY, USA, 2012).

Kullback, S. and R. A. Leibler, "On information and sufficiency", Ann. Math. Statist. **22**, 1, 79–86, URL https://doi.org/10.1214/aoms/1177729694 (1951).

Lakshminarayanan, B., A. Pritzel and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles", in "Proceedings of the 31st International Conference on Neural Information Processing Systems", NIPS'17, p. 64056416 (Curran Associates Inc., Red Hook, NY, USA, 2017).

Lattice Semiconductor, URL http://www.latticesemi.com/en/Products (2021).

Lecun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE **86**, 11, 2278–2324 (1998).

LeCun, Y. and C. Cortes, "MNIST handwritten digit database", URL http://yann.lecun.com/exdb/mnist/ (2010).

Lee, M., K. Hwang, J. Park, S. Choi, S. Shin and W. Sung, "Fpga-based low-power speech recognition with recurrent neural networks", in "2016 IEEE International Workshop on Signal Processing Systems (SiPS)", pp. 230–235 (2016).

Leibig, C., V. Allken, M. S. Ayhan, P. Berens and S. Wahl, "Leveraging uncertainty information from deep neural networks for disease detection", Nature URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5736701/ (2017).

Leshno, M., V. Y. Lin, A. Pinkus and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function", Neural Networks **6**, 6, 861 – 867, URL http://www.sciencedirect.com/science/article/pii/S0893608005801315 (1993).

Li, H., Z. Lin, X. Shen, J. Brandt and G. Hua, "A convolutional neural network cascade for face detection", in "2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", pp. 5325–5334 (2015).

Li, S., C. Wu, H. Li, B. Li, Y. Wang and Q. Qiu, "Fpga acceleration of recurrent neural network based language model", in "Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines", FCCM '15, pp. 111–118 (IEEE Computer Society, Washington, DC, USA, 2015), URL http://dx.doi.org/10.1109/FCCM.2015.50.

Lichman, M. *et al.*, "Uci machine learning repository", (2013).

Lin, M., Q. Chen and S. Yan, "Network in network", (2013).

Liu, H. J., *Archipelago-An Open Source FPGA with Toolflow Support*, Master's thesis, EECS Department, University of California, Berkeley, URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-43.html (2014).

Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "Ssd: Single shot multibox detector", (2016), URL http://arxiv.org/abs/1512.02325, to appear.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach", (2019).

Lu, Z., H. Pu, F. Wang, Z. Hu and L. Wang, "The expressive power of neural networks: A view from the width", in "Advances in Neural Information Processing Systems 30", edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, pp. 6231–6239 (Curran Associates, Inc., 2017), URL http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf.

Mah Ung, G., "Pcworld", URL https://www.pcworld.com/article/3072256 (2018).

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, "Equation of state calculations by fast computing machines", The Journal of Chemical Physics **21**, 6, 1087–1092, URL http://link.aip.org/link/?JCP/21/1087/1 (1953).

Mukhoti, J. and Y. Gal, "Evaluating bayesian deep learning methods for semantic segmentation", CoRR **abs/1811.12709**, URL http://arxiv.org/abs/1811.12709 (2018).

Muller, J.-M., *Elementary Functions: Algorithms and Implementation* (Birkhauser, 2005).

Muroga, S., *Threshold Logic and its Applications* (Wiley-Interscience New York, 1971).

Norman P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit", in "Proceedings of the 44th Annual International Symposium on Computer Architecture", ISCA '17, pp. 1–12 (ACM, New York, NY, USA, 2017), URL `http://doi.acm.org/10.1145/3079856.3080246`.

Nurvitadhi, E., Jaewoong Sim, D. Sheffield, A. Mishra, S. Krishnan and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic", in "2016 26th International Conference on Field Programmable Logic and Applications (FPL)", pp. 1–4 (2016).

Oladyshkin, S. and W. Nowak, "Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion", Reliability Engineering and System Safety **106**, 179–190 (2012).

Park, E., S. Yoo and P. Vajda, "Value-aware quantization for training and inference of neural networks", (2018a).

Park, J., M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. S. Khudia, J. Law, P. Malani, A. Malevich, N. Satish, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. M. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo and M. Smelyanskiy, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications", CoRR **abs/1811.09886**, URL `http://arxiv.org/abs/1811.09886` (2018b).

Rahman, S., "Wiener-hermite polynomial expansion for multivariate gaussian probability measures", Journal of Mathematical Analysis and Applications **454**, 1, 303 – 334, URL `http://www.sciencedirect.com/science/article/pii/S0022247X17304250` (2017).

Rasmussen, C. E. and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)* (The MIT Press, 2005).

Ren, S., K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in "Advances in Neural Information Processing Systems 28", edited by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett, pp. 91–99 (Curran Associates, Inc., 2015), URL `https://proceedings.neurips.cc/paper/2015`.

Robinson, T. D., M. S. Eldred, K. E. Willcox and R. Haimes, "Surrogate-based optimization using multifidelity models with variable parameterization and corrected space mapping", AIAA Journal **46**, 11, 2814–2822, URL `https://doi.org/10.2514/1.36043` (2008).

Rose, J., J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson and J. Anderson, "The VTR project: Architecture and cad for fpgas from verilog to routing", in "Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays", FPGA '12, pp. 77–86 (2012).

Rosenblatt, F., "The perceptron: A probabilistic model for information storage and organization in the brain.", Psychological Review (1958).

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", International Journal of Computer Vision (IJCV) **115**, 3, 211–252 (2015).

Rybalkin, V., A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn and M. Blott, "FINN-L: library extensions and design trade-off analysis for variable precision LSTM networks on fpgas", in "28th International Conference on Field Programmable Logic and Applications, FPL 2018, Dublin, Ireland, August 27-31, 2018", pp. 89–96 (2018), URL https://doi.org/10.1109/FPL.2018.00024.

Rybalkin, V., N. Wehn, M. R. Yousefi and D. Stricker, "Hardware architecture of bidirectional long short-term memory neural network for optical character recognition", in "Proceedings of the Conference on Design, Automation & Test in Europe", DATE '17, pp. 1394–1399 (European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 2017), URL http://dl.acm.org/citation.cfm?id=3130379.3130707.

Saltelli, A. and I. M. Sobol', "About the use of rank transformation in sensitivity analysis of model output", Reliability Engineering and System Safety **50**, 3, 225 – 239, URL http://www.sciencedirect.com/science/article/pii/0951832095000992 (1995).

Schäfer, A. M. and H. G. Zimmermann, "Recurrent neural networks are universal approximators", in "Proceedings of the 16th International Conference on Artificial Neural Networks - Volume Part I", ICANN06, p. 632640 (Springer-Verlag, Berlin, Heidelberg, 2006), URL https://doi.org/10.1007/11840817_66.

Shin, D., J. Lee, J. Lee and H. Yoo, "14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks", in "2017 IEEE International Solid-State Circuits Conference (ISSCC)", pp. 240–241 (2017).

Sim, H. and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks", in "Proceedings of the 54th Annual Design Automation Conference 2017", DAC '17, pp. 29:1–29:6 (ACM, New York, NY, USA, 2017), URL http://doi.acm.org/10.1145/3061639.3062290.

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", (2014).

Sobolá, I. M., "Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates", Math. Comput. Simul. **55**, 13, 271280, URL https://doi.org/10.1016/S0378-4754(00)00270-6 (2001).

Soize, C. and R. Ghanem, "Physical systems with random uncertainties: Chaos representations with arbitrary probability measure", SIAM Journal of Scientific Computing **26**, 2 (2004).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", Journal of Machine Learning Research **15**, 56, 1929–1958, URL `http://jmlr.org/papers/v15/srivastava14a.html` (2014).

Srivastava, N., E. Mansimov and R. Salakhutdinov, "Unsupervised learning of video representations using lstms", in "Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37", ICML'15, pp. 843–852 (JMLR.org, 2015), URL `http://dl.acm.org/citation.cfm?id=3045118.3045209`.

Stollenga, M. F., W. Byeon, M. Liwicki and J. Schmidhuber, "Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation", in "Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2", NIPS'15, pp. 2998–3006 (MIT Press, Cambridge, MA, USA, 2015), URL `http://dl.acm.org/citation.cfm?id=2969442.2969574`.

Sudret, B., "Uncertainty propagation and sensitivity analysis in mechanical models – contributions to structural reliability and stochastic spectral methods", (2007).

Sudret, B., "Global sensitivity analysis using polynomial chaos expansions", Reliability Engineering and System Safety **93**, 7, 964–979 (2008).

Sudret, B. and A. D. Kiureghian, "Stochastic finite elements and reliability: a state-of-the-art report", in "Technical Report UCB/SEMM-2000/08, University of California, Berkeley", (2000).

Sundermeyer, M., H. Ney and R. Schlter, "From feedforward to recurrent lstm neural networks for language modeling", IEEE/ACM Transactions on Audio, Speech, and Language Processing **23**, 3, 517–529 (2015).

Sutskever, I., O. Vinyals and Q. V. Le, "Sequence to sequence learning with neural networks", in "Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2", NIPS'14, pp. 3104–3112 (MIT Press, Cambridge, MA, USA, 2014), URL `http://dl.acm.org/citation.cfm?id=2969033.2969173`.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the inception architecture for computer vision", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 2818–2826 (2015).

Szegedy, C., Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions", in "2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", pp. 1–9 (2015).

Tfekci, P., “Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods”, International Journal of Electrical Power and Energy Systems **60**, 126 – 140, URL http://www.sciencedirect.com/science/article/pii/S0142061514000908 (2014).

Toshev, A. and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks”, in “2014 IEEE Conference on Computer Vision and Pattern Recognition”, pp. 1653–1660 (2014).

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser and I. Polosukhin, “Attention is all you need”, in “Proceedings of the 31st International Conference on Neural Information Processing Systems”, NIPS17, p. 60006010 (Curran Associates Inc., Red Hook, NY, USA, 2017).

Vrudhula, S., J. Wang and P. Ghanta, “Hermite polynomial based interconnect analysis in the presence of process variations”, IEEE Transactions on Computer Aided Design **25**, 10, 2001–20011 (2006).

Wagle, A., G. Singh, J. Yang, S. Khatri and S. Vrudhula, “Threshold logic in a flash”, in “IEEE International Conference on Computer Design (ICCD)”, (2019).

Wang, N. and D.-Y. Yeung, “Learning a deep compact image representation for visual tracking”, in “Advances in Neural Information Processing Systems 26”, edited by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger, pp. 809–817 (Curran Associates, Inc., 2013), URL https://proceedings.neurips.cc/paper/2013.

Wang, S., Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang and Y. Liang, “C-lstm: Enabling efficient lstm using structured compression techniques on fpgas”, in “Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays”, FPGA ’18, pp. 11–20 (ACM, New York, NY, USA, 2018), URL http://doi.acm.org/10.1145/3174243.3174253.

Wang, Z., J. Lin and Z. Wang, “Accelerating recurrent neural networks: A memory-efficient approach”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **25**, 10, 2763–2775 (2017).

Wiener, N., “The homogeneous chaos”, American Journal of Mathematics **60**, 4, 897–936, URL http://www.jstor.org/stable/2371268 (1938).

Xiu, D. and G. E. Karniadakis, “The wiener–askey polynomial chaos for stochastic differential equations”, SIAM J. Sci. Comput. **24**, 2, 619644, URL https://doi.org/10.1137/S1064827501387826 (2002).

Xu, Y., Y. Wang, A. Zhou, W. Lin and H. Xiong, “Deep neural network compression with single and multiple level quantization”, CoRR **abs/1803.03289**, URL http://arxiv.org/abs/1803.03289 (2018).

Yang, Z., Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding", in "Advances in Neural Information Processing Systems 32", pp. 5753–5763 (Curran Associates, Inc., 2019), URL https://proceedings.neurips.cc/paper/2019.

Yazdanbakhsh, A., A. T. Elthakeb, P. Pilligundla, F. Mireshghallah and H. Esmaeilzadeh, "Releq: A reinforcement learning approach for deep quantization of neural networks", (2018), URL https://arxiv.org/pdf/1811.01704.pdf.

Yu, L., S. Saxena, C. Hess, A. Elfadel, D. Antoniadis and D. Boning, "Remembrance of transistors past: Compact model parameter extraction using bayesian inference and incomplete new measurements", in "2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)", pp. 1–6 (2014).

Zhang, Z., T. A. El-Moselhy, I. M. Elfadel and L. Daniel, "Stochastic testing method for transistor-level uncertainty quantification based on generalized polynomial chaos", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **32**, 10, 1533–1545 (2013).

Zhang, Z., X. Yang, I. V. Oseledets, G. E. Karniadakis and L. Daniel, "Enabling high-dimensional hierarchical uncertainty quantification by anova and tensor-train decomposition", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **34**, 1, 63–76 (2015).

Zhou, D.-X., "Universality of deep convolutional neural networks", Applied and Computational Harmonic Analysis **48**, 2, 787 – 794, URL http://www.sciencedirect.com/science/article/pii/S1063520318302045 (2020).

APPENDIX A

POLYNOMIAL CHAOS EXPANSION

A.1 A Brief History of Polynomial Chaos Theory

When analyzing a physical system, a key task is to understand the relationship between the system's inputs and outputs/response. In the event that the inputs are represented by random variables, stochastic processes or random fields, the system response will also be a random quantity. In general, a system under consideration is given in some form that represents an implicit relation $F(X, Y, \xi)$, where X, Y and ξ are the set of inputs, set of outputs and a set of parameters that are subject to variations. Each of these quantities are assumed to be vectors of arbitrary dimensions.

One approach is to approximate the system's response by replacing the implicit relation $F(X, Y, \xi)$ with an explicit function $\widetilde{Y}(X, \xi)$. Then, large number of samples of ξ would be generated and $\widetilde{Y}(X, \xi)$ would be computed. In the stochastic case, the random variables Y (assume finite variance) can be expressed as an infinite series of multi-variate orthogonal polynomials of all (infinite) orders, referred to as *polynomial chaos expansion* (PCE) Ghanem and Spanos (1991); Soize and Ghanem (2004); Augustin *et al.* (2008).

PCE is a representation of a 2nd order stochastic process as a multivariate orthogonal polynomial over an infinite dimensional Hilbert space. In other words, for any random process $Y \in L^2$, the PCE representation for Y exhibits L^2 convergence, i.e. the first two moments of the expansion converge to the mean and variance of Y in limit. Based on the Cameron and Martin theorem Cameron and Martin (1947), such an expansion converges in the L^2 sense for any arbitrary stochastic process with finite second moment.

Definition A.1.1 (Finite Second Moment)**.** For a given probability space $(\Omega, \mathfrak{A}, P)$, the set of random variables $X : \Omega \to R$ that satisfy $E(X^2) < \infty$ have *finite second moment* and are in L^2.

Definition A.1.2 (Hilbert Space)**.** Let $\mathcal{H}$ be a vector space over some field $\mathcal{F}$ with inner product $\langle f, g \rangle$ $f, g \in \mathcal{H}$ defined. The norm in $\mathcal{H}$ is $\|f\| = \sqrt{\langle f, f \rangle}$, and the metric is $d(f, g) = \|f - g\|$. $\mathcal{H}$ is called a Hilbert space if it is complete as a metric space.

Definition A.1.3 (Orthogonality)**.** Two elements, x and y of an inner product space are said to be orthogonal if $\langle x, y \rangle = 0$. In addition, if $\|x\| = \|y\| = 1$, they are orthonormal.

The classical PCE was originated from the Wiener's work Wiener (1938) and is known as *Wiener-Hermite expansion*. It represents a random variable with an infinite series of Hermite polynomials in independent Gaussian random variables as shown in equation A.1.

$$Y(\xi) = \sum_{i=0}^{\infty} c_i H_i(\xi), \tag{A.1}$$

where c_i are the coefficients of the Hermite polynomial basis functions, i.e. $H_i(\xi)$. In practice, this infinite sum is truncated to a limited number of basis functions and is referred to as approximated PCE, as expressed in A.2.

$$\widetilde{Y}(\xi) = \sum_{i=0}^{M} c_i \Phi_i(\xi), \tag{A.2}$$

where $\Phi_i(\xi)$ are the basis functions (determined based on a selected method) and c_i are the coefficients that need to be solved. The coefficients c_i can be obtained by Galerkin projection method. This method determines the function $\widetilde{Y}$ in such a way that the error $(Y - \widetilde{Y})$ is orthogonal to the space where Y belongs to. This is known as *projection* and is also referred to as the *Galerkin*'s method. A set of Galerkin methods is an example of projection methods with the property of orthogonality principle.

Theorem A.1.1 (Hilbert Projection Theorem)**.** Given a subspace W of estimators within a Hilbert space V and an element $v \in V$, a vector element $w \in W$ achieves minimum mean squared error among all the elements in W if and only if:

$$E\{(v - w)y^T\} = 0 \quad for\ all\ y \in W \tag{A.3}$$

As an example, Figure A.1 illustrates an optimum projection of vector $v \in V$ into a subspace W, in which the estimation error $(v - w)$ is orthogonal to subspace W that provides the best approximation.

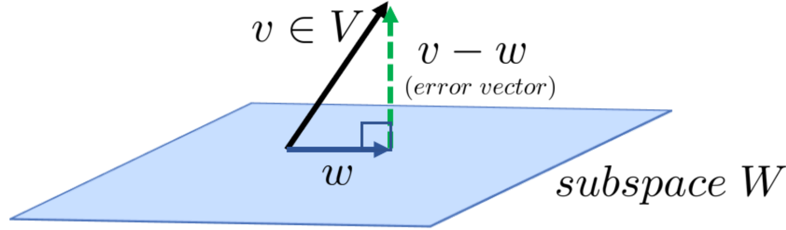


Figure A.1: An Optimum Projection of Vector $v \in V$ into a Subspace W.

Cameron and Martin Cameron and Martin (1947) proved convergence of the *Wiener-Hermite* PCE for a general square integrable process. Cameron and Martin-Cameron and Martin (1947) proved that the polynomial representation has optimal L^2 convergence (i.e. convergence in probability) to the actual process.

Theorem A.1.2 (Cameron-Martin Theorem)**.** For any functional f in a Hilbert measure space $(\mathcal{X}, \mathcal{M}, \mu)$, there exist a set of polynomials ϕ_i and constants a_i such that:

$$\lim_{N \to \infty} \int_{\mathcal{X}} (f(x) - \hat{f}_N(x))^2 d\mu(x) = 0, \tag{A.4}$$

where $\hat{f}_N(x) = \sum_{i=0}^{N} a_i \phi_i(x)$ and a_i is obtained from the Galerkin projection $a_i := \frac{\langle f, \phi_i \rangle}{||\phi_i^2||}$.

A.2 Generalized Polynomial Chaos

PCE is not restricted to Gaussian processes and has been generalized for other standard distributions. Xiu and Karniadakis Xiu and Karniadakis (2002) introduced generalized polynomial chaos (gPCE) involving non-Gaussian random parameters. The optimal basis is dependent on the underlying distribution of the basic random variables, i.e. the Hermite polynomials are replaced by the sequence of polynomials orthogonal with respect to the probability distribution of the basic random variables ($\boldsymbol{\xi}$). Table A.1 outlines the optimal orthogonal basis functions for some common distributions. If the distribution is unknown, it is typically assumed as Gaussian.

Table A.1: Optimal Polynomials for Various Continuous Distributions Eldred and Burkardt (2012).

Distribution	Density Function	Polynomial	Support Range
Uniform	$\frac{1}{2}$	Legendre $P_n(x)$	[-1,1]
Exponential	e^{-x}	Laguerre $L_n(x)$	$[0, \infty]$
Gaussian	$\frac{1}{\sqrt{2\pi}}e^{\frac{-x^2}{2}}$	Hermite $He_n(x)$	$[-\infty, \infty]$
Beta	$\frac{(1-x)^a(1+x)^\beta}{2^{\alpha+\beta+1}B(\alpha+1,\beta+1)}$	Jacobi $P_n^{(\alpha,\beta)}(x)$	[-1, 1]
Gamma	$\frac{x^\alpha e^{-x}}{\Gamma(\alpha+1)}$	Generalized Laguerre $L_n^{(\alpha)}(x)$	$[0, \infty]$

Based on the underlying distribution of $\boldsymbol{\xi}$, the optimal polynomials for the gPC can be determined. Table A.1 presents the link between a few examples of continuous distributions and their corresponding optimal polynomials.

If ξ_is are known to be Gaussian, then the optimal polynomial for the gPC is Hermite polynomial as shown in Table A.1. A multivariate Hermite polynomial ($He_n(\boldsymbol{\xi})$) of order n is expressed in Equation A.5.

$$He_n(\boldsymbol{\xi}) = (-1)^n e^{\frac{\boldsymbol{\xi}^2}{2}} \frac{\partial^n}{\partial \boldsymbol{\xi}^n} e^{-\frac{\boldsymbol{\xi}^2}{2}} \tag{A.5}$$

For example, consider $\boldsymbol{\xi}$ to be a two-dimensional vector, consisting of Gaussian random variables ξ_1 and ξ_2. Without loss of generality, the random variables are considered as standard Gaussian with zero mean and unit variance. The Hermite polynomials up to the 3rd degree are computed based on Equation A.5 and the results are shown in Equations A.6-A.9.

$$0^{th} order: \; He_0(\boldsymbol{\xi}) = 1 \tag{A.6}$$

$$1^{st} order: \; He_1(\boldsymbol{\xi}) = [\xi_1 \; , \; \xi_2] \tag{A.7}$$

$$2^{nd} order: \; He_2(\boldsymbol{\xi}) = [\xi_1^2 - 1 \; , \; \xi_1\xi_2 \; , \; \xi_2^2 - 1 \;] \tag{A.8}$$

$$3^{rd} order: \; He_3(\boldsymbol{\xi}) = [\xi_1^3 - 3\xi_1 \; , \; \xi_1^2\xi_2 - \xi_2 \; , \; \xi_1\xi_2^2 - \xi_1 \; , \; \xi_2^3 - 3\xi_2] \tag{A.9}$$

Ernst et al. Ernst *et al.* (2012) proved that the L^2 convergence holds true for general random processes as long as the their second moment is finite.

A.2.1 General Polynomial Chaos with Finite Series

Consider the mth order gPCE which contains all the polynomials with the total degree less than or equal to $m \in N$. This is denoted by $y_m(\boldsymbol{\xi})$, which is an approximation to the original model with infinite number of polynomials ($y(\boldsymbol{\xi})$). Sharif Rahman Rahman (2017) showed that the probabilistic characteristics of $y(\boldsymbol{\xi})$, including its first two moments and p.d.f, if it exists, can be estimated from the statistical properties of the truncated gPCE, $y_m(\boldsymbol{\xi})$. Particularly, Sharif Rahman Yazdanbakhsh *et al.* (2018) proved that the mean of these two models are the same as the zero degree expansion coefficient (i.e. c_0) and are independent of m. This is expressed in Equation A.10.

$$E[y_m(\boldsymbol{\xi})] = E[y(\boldsymbol{\xi})] = c_0 \tag{A.10}$$

Applying the expectation operator on $[y_m(\boldsymbol{\xi}) - c_0]^2$ and $[y(\boldsymbol{\xi}) - c_0]^2$ for computing the variance results in Equations A.11 and A.12 Rahman (2017).

$$var[y_m(\boldsymbol{\xi})] = \sum_{1 \leq |j| \leq m} c_j^2 \tag{A.11}$$

$$var[y(\boldsymbol{\xi})] = \sum_{1 \leq |j| \leq \infty} c_j^2 \tag{A.12}$$

Hence, the approximation of the variance (i.e. $var[y_m(\boldsymbol{\xi})]$) approaches the exact variance $var[y(\boldsymbol{\xi})]$ as $m \to \infty$. It is worth mentioning that the p.d.f of $y(\boldsymbol{\xi})$ can also be estimated with $y_m(\boldsymbol{\xi})$.

A.3 Arbitrary Polynomial Chaos

The drawback of having to determine or assume a distribution function of the random variables was eliminated by the method proposed by Oladyshkin et al. Oladyshkin and Nowak (2012). In this method, referred to as *arbitrary polynomial chaos* (aPC), the coefficients of the basis polynomials are given as explicit functions of the moments of the random variables. In practice, sample moments are used, making the method completely data-driven. This eliminates the need to assume any particular distribution and also eliminates the computationally intensive task of solving integrals for determining the coefficients. aPC involves two steps. First, a set of multivariate orthogonal basis polynomials in the random variables (rvs) ξ is constructed. The coefficients of each of the basis polynomials are functions of the moments of the rvs ξ. They are not related to the output $Y(\xi)$, and hence computed once given the data ξ. Next, the output $Y(\xi)$ is expressed as a weighted linear combination of the basis polynomials. The weights or coefficients of this series are estimated in the traditional way – by minimizing the mean-squared error between the polynomial values and the training data.

Let $\xi = \{\xi_1, \xi_2, ..., \xi_N\}$. $Y(\xi)$ is represented as

$$Y(\xi) = \sum_{i=1}^{M} c_i \Phi_i(\xi). \tag{A.13}$$

$\Phi_i(\xi)$s are the orthonormal multivariate basis polynomials, and the c_i are the unknown coefficients to be estimated. Their number is $M = (N + d)!/(N!d!)$, where d is the degree of the expansion. $\Psi_i(\xi)$s are constructed by only using the moments of the process variables and no other statistical information including the underlying distributions is required. The sum in Equation A.13 involves an infinite number of polynomial terms but in practice, the sum is truncated to a finite sum and is an important step. The truncation needs to avoid removing too many polynomial terms or adding several extra terms. These would lead to underfitting and overfitting, respectively and the model would not be generalizable.

The characteristic statistical quantities of $Y(\xi)$ can be evaluated directly from the coefficients c_i. The mean (μ) and variance (σ^2) of Y are computed using Equation A.14.

$$\mu_Y = c_0, \quad \text{and} \quad \sigma_Y^2 = \sum_{i=1}^{M} c_i^2. \tag{A.14}$$

Assuming the input parameters are independent, the multivariate basis functions can be constructed as in Equation A.15.

$$\Phi_i(\xi) = \prod_{j=1}^{N} P_j^{(\alpha_j^i)}(\xi), \tag{A.15}$$

$$\sum_{j=1}^{N} \alpha_j^i \leq M, \quad i = 1, 2..., N, \tag{A.16}$$

where α is an $M \times N$ matrix, which contains the corresponding degree for process variable number j in the expansion term k. Hence, α_j^i is an index that contains the

combinatoric information on how to enumerate all possible products of individual univariate basis functions. $P^{(k)}$ are the orthonormal univariate polynomial basis of order k as shown in Equation A.17, where p_i^k are their corresponding coefficients.

$$P^k(\xi) = \sum_{i=0}^{k} p_i^k \xi^i, \quad k = 0, 1, ..., d. \tag{A.17}$$

Based on the Equation-14 in Oladyshkin and Nowak (2012), the system of linear equations can be written as in Equations A.18-A.20 for 0th to 2nd degree. The μ and p are the raw moment and the coefficient of the uni-variate polynomials P, respectively. The equations for higher orders are eliminated for brevity.

$$[1] \times \left[p_0^{(0)}\right] = [1] \qquad (0^{th} degree) \tag{A.18}$$

$$\begin{bmatrix} \mu_0 & \mu_1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} p_0^{(1)} \\ p_1^{(1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad (1^{st} degree) \tag{A.19}$$

$$\begin{bmatrix} \mu_0 & \mu_1 & \mu_2 \\ \mu_1 & \mu_2 & \mu_3 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} p_0^{(2)} \\ p_1^{(2)} \\ p_2^{(2)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad (2^{nd} degree) \tag{A.20}$$

By solving the linear equations A.18-A.20 for normalized distribution of data with zero mean and unit variance, the coefficients of the uni-variate polynomials for 0th degree up to the 2nd degree can be computed as in Equations A.21-A.23.

$$p_0^0 = 1 \tag{A.21}$$
$$p_0^1 = 0, \ p_1^1 = 1 \tag{A.22}$$
$$p_0^2 = -1, \ p_1^2 = -\mu_3, \ p_2^2 = 1 \tag{A.23}$$

Equations A.24-A.26 express the uni-variate polynomials up to the 2nd order by plugging in the coefficients p in equation A.17.

$$P^{(0)}(\xi) = 1 \tag{A.24}$$
$$P^{(1)}(\xi) = \xi \tag{A.25}$$
$$P^{(2)}(\xi) = \xi^2 - \mu_3\xi - 1 \tag{A.26}$$

In general, the system of linear equations can be written in a matrix form as in Equation A.27. To compute the kth order $P(\xi)$, moments up to $2k - 1$ are required.

$$\begin{bmatrix} \mu_0 & \mu_1 & \dots & \mu_k \\ \mu_1 & \mu_2 & \dots & \mu_{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ \mu_{k-1} & \mu_k & \dots & \mu_{2k-1} \\ 0 & 0 & \dots & 1 \end{bmatrix} \times \begin{bmatrix} p_0^{(k)} \\ p_1^{(k)} \\ \vdots \\ p_{k-1}^{(k)} \\ p_k^{(k)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{A.27}$$

The multivariate basis polynomials (Φ) can be obtained by taking the cross product of univariate basis functions. After computing the coefficients (c_i) using least squares, the response function can be modeled using Equation A.13. The coefficients are computed as a one time cost and remain unchanged for evaluation, hence the mean and variance. To estimate the uncertainty in the response, the predicted output is compared against the mean and variance of $Y(\xi)$.

As a concrete example, consider a system with two input variables, denoted by ξ_1 and ξ_2, which are subject to variability. The coefficients of the 0th and 1st order polynomials for each variable is expressed in Equation A.28. The second order univariate polynomials whose coefficients are functions of the moments (μ) are expressed in Equation A.29.

$$p_0^{(2)} = -1,\ p_1^{(2)} = -\mu_3,\ p_2^{(2)} = 1 \tag{A.28}$$

$$P^{(0)}(\xi) = 1,\ P^{(1)}(\xi) = \xi,\ P^{(2)}(\xi) = \xi^2 - \mu_3\xi - 1 \tag{A.29}$$

The multivariate polynomial basis for a $2nd$ order model are obtained by taking the cross product of the univariate polynomials as shown in Equation A.30 to compute the output based on Equation A.31. There are six basis polynomials and six unknown coefficients (c_i), which are solved using least squares as in Equation A.32, where S is the number of samples.

$$\Psi(\xi_1, \xi_2) = \{1,\ \xi_1,\ \xi_2,\ \xi_1\xi_2, \xi_1^2 - \mu_3^1\xi_1 - 1,\ \xi_2^2 - \mu_3^2\xi_2 - 1\}, \tag{A.30}$$

$$Y(\boldsymbol{\xi}) = \sum_{i=1}^{\binom{N+d}{d}} c_i \Psi_i(\boldsymbol{\xi}). \tag{A.31}$$

$$J(c) = \frac{1}{2}\sum_{s=1}^{S} \{Y_s^{obs} - Y_s(\boldsymbol{\xi})\}^2 \tag{A.32}$$

If the underlying distribution of the random variables are known, then the aPC and gPC expansion are the same. To demonstrate this, consider the random variables to be multivariate Gaussian and $\boldsymbol{\xi} = [\xi_1, \xi_2, ..., \xi_n]$.

Table A.2 presents the central and non-central moments of a normal r.v up to the 5th order. Similar to the gPC case, the central moments are used in here as well. The odd moments are zero and the even moments are functions of sigma.

Table A.2: Central and Non-central Moments of a Normal Random Variable up to Order 5.

order	Non-central Moment	Central Moment
1	μ	0
2	$\mu^2 + \sigma^2$	σ^2
3	$\mu^3 + 3\mu\sigma^2$	0
4	$\mu^4 + 6\mu^2\sigma^2 + 3\sigma^4$	$3\sigma^4$
5	$\mu^5 + 10\mu^3\sigma^2 + 15\mu\sigma^4$	0

By solving the system of linear equations shown in Equation A.27, the following uni-variate polynomials are generated.

$$P^{(0)}(\xi_i) = 1 \tag{A.33}$$
$$P^{(1)}(\xi_i) = \xi_i \tag{A.34}$$
$$P^{(2)}(\xi_i) = \xi_i^2 - \mu_3\xi_i - 1 \tag{A.35}$$

$$\begin{aligned} P^{(3)}(\xi_i) =& \xi_i^3 + \frac{-(\mu_3 - \mu_5 + \mu_3\mu_4)}{\mu_3^2 - \mu_4 + 1}\xi_i^2 + \frac{-(-\mu_3^2 + \mu_5\mu_3 - \mu_4^2 + \mu_4)}{\mu_3^2 - \mu_4 + 1}\xi_i + \\ & \frac{-(\mu_3^3 - 2\mu_4\mu_3 + \mu_5)}{\mu_3^2 - \mu_4 + 1} \end{aligned} \tag{A.36}$$

Based on the central moments for the Hermite polynomials presented in Table A.2, the uni-variate polynomials are simplified as follows.

$$P^{(0)}(\xi_i) = 1 \tag{A.37}$$
$$P^{(1)}(\xi_i) = \xi_i \tag{A.38}$$
$$P^{(2)}(\xi_i) = \xi_i^2 - 1 \tag{A.39}$$
$$P^{(3)}(\xi_i) = \xi_i^3 - 3\xi_i \tag{A.40}$$

For the example above for two random variables ξ_1 and ξ_2, the uni-variate polynomials are as follows.

$$P^{(0)}(\boldsymbol{\xi}) = [1\ ,\ 1] \tag{A.41}$$
$$P^{(1)}(\boldsymbol{\xi}) = [\xi_1\ ,\ \xi_2] \tag{A.42}$$
$$P^{(2)}(\boldsymbol{\xi}) = [\xi_1^2 - 1\ ,\ \xi_2^2 - 1] \tag{A.43}$$
$$P^{(3)}(\boldsymbol{\xi}) = [\xi_1^3 - 3\xi_1\ ,\ \xi_2^3 - 3\xi_2] \tag{A.44}$$

Since the uni-variates are orthogonal then the multivariate polynomials are the products of uni-variate polynomials of all combinations, which are as follows.

$$0^{th} order: \ \Phi_0(\boldsymbol{\xi}) = 1 \tag{A.45}$$

$$1^{st} order: \ \Phi_1(\boldsymbol{\xi}) = [\xi_1 \ , \ \xi_2] \tag{A.46}$$

$$2^{nd} order: \ \Phi_2(\boldsymbol{\xi}) = [\xi_1^2 - 1 \ , \ \xi_1\xi_2 \ , \ \xi_2^2 - 1 \ ] \tag{A.47}$$

$$3^{rd} order: \ \Phi_3(\boldsymbol{\xi}) = [\xi_1^3 - 3\xi_1 \ , \ \xi_1^2\xi_2 - \xi_2 \ , \ \xi_1\xi_2^2 - \xi_1 \ , \ \xi_2^3 - 3\xi_2] \tag{A.48}$$

Hence, the multi-variate for the gPC and aPC are equal when the r.v.s are normal, as shown in Equations A.6-A.9 and Equations A.45-A.48 .