DESIGN SPACE EXTRAPOLATION AND INVERSE DESIGN USING MACHINE LEARNING

A Proposal for Doctoral Dissertation
Presented to
The Academic Faculty

By

Osama Waqar Bhatti

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering
College of Engineering

Georgia Institute of Technology

December 2022

DESIGN SPACE EXTRAPOLATION AND INVERSE DESIGN USING MACHINE LEARNING

Thesis committee:

Dr. Madhavan Swaminathan, Advisor
Department of Electrical and Computer Engineering, Department of Material Science and Engineering
Georgia Institute of Technology

Dr. Sung-kyu Lim
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Saibal Mukhopadhyay
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Arijit Raychowdhury
Department of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Suresh Sitaraman
Department of Mechanical Engineering
Georgia Institute of Technology

Date approved: October 31, 2022

To my loving mama and papa, without whom this would not have been possible.

ACKNOWLEDGMENTS

I was told I have a knack for solving harder problems easier and making simple problems harder. That approach is what led me strive for a doctoral degree. I would like to take a moment and thank the wonderful people who have helped directly and indirectly in my path to completing the dissertation.

Foremost, I would like to thank my advisor, Prof. Madhavan Swaminathan who constantly instilled in me the drive to perform research tasks with success and provided me with the opportunity to think outside the box. I have seen him in multiple roles, as a professor, a researcher, an academic advisor and more importantly an idea communicator. Every interaction with him is a learning opportunity.

I would like to thank my committee members: Dr. Sung-kyu Lim, Dr. Arijit Raychowdary, Dr. Saibal Mukhopadhyay, Dr. Suresh Sitaraman for their time and valuable feedback to improve my research. I would like to thank the amazing people I had a chance to take a class from: Prof. Andrew F. Peterson, Prof. Dhruv Bhatra and Prof. Hua Wang.

I also would like to thank the support and feedback that I received from industry members of Center for Advanced Electronics through Machine Learning (CAEML). Especially to Dr. Jose Hejase, Dr. Jing Wang, Sunil Sudhakaran from Nvidia, Dr. Dale Becker, Dr. Xianbo Yang from IBM, and Dr. Kemal Aygun from Intel, for their invaluable insights and feedback to constantly improve my research to address real-world problems. I am also grateful for the wonderful post-docs I have had a chance to work with: Dr. Mourad Larbi, Dr. Kallol Roy, Dr. Nikita Ambasana and Dr. Rahul Kumar. I would also like to pay my regards to my baccalaureate degree advisor Dr. Syed Ali Hassan because of whom I got the drive to start a doctorate program.

Many of the results in this thesis would not have been possible without the help and motivation of my friends in the lab: Dr. Huan Yu, Dr. Sridhar Sivapurapu, Dr. Hakki Mert Torun, Dr. Majid Ahadi, Mutee ur Rehman, Serhat Erdogan, Claudio Alvarez, Se-

unghyup Han, Lakshmi Narasimha, Xiaofan Jia, Xingchen Li, Nahid Amoli, Eric Huang, Kai-Qi Huang, Venkatesh Avula, Oluwaseyi Akinwande, Prahalad Murali, Pragna Bhaskar, Pavithra Kuppakone and Pratik Nimbalkar. Thank you for all the joyful memories.

Outside of the academia life, I am also grateful to all the friends I have had a chance to meet which kept life in Atlanta interesting: Mahmut Burak Okuducu, Ayesha Shahid, Zulfiqar Zaidi, Wajahat Latif, Ahsan Cheema, Muhammad Ahmad Mustafa, Nazar Abbas and Rehab Maqsood.

I was also fortunate enough to enjoy the support and company of my friends from college: Haris Suhail, Uzair Akber, Akber Raza, Mani, Saad, Dogar and Bhalu. You guys have been there virtually even though were distributed everywhere on the globe achieving great things.

I owe my deepest gratuitude to my family: my mother Sabah and my father Waqar Bhatti. It was you who taught me to get up everytime we fall and to keep cracking the problem. I must also thank my partner in crime my brother Hamza, for always being there as my punching bag, always showing me there is a better way to do things. He is younger than me but he is intellectually and emotionally a natural inspiration to learn from. Making our lives even richer is my sister-in-law Marryam who has been the most fun-loving yet extremely capable person in regards to her profession and personal life. I also owe the completion of this accomplishment to my Dada abu (paternal grandfather) Rasheed Saqi who has deeply impacted my life values of hard work and honesty and my Nana abu (maternal grandfather) Aman Ullah who has shown me that it is always more important to be kind than right.

This thesis was funded in part by the National Science Foundation under Grant No. CNS 16-24810- Center for Advanced Electronics through Machine Learning(CAEML), 3D Packaging Research Center (PRC) at Georgia Tech.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

The objective of the proposed research is to investigate machine learning techniques for power delivery, signal integrity and EM problems. Two broad design strategies have been analyzed. Often one needs to predict the structure behavior outside the range of simulations. This work deals with extrapolation in two domains. (1) We propose HilbertNet for complex-valued causal extrapolation of frequency responses. The proposed architecture accurately predicts the out-of-band frequency response by modelling the temporal correlations between in-band frequency samples using specialized recurrent neural networks. The proposed architecture has been applied to a wide variety of applications including transmission lines, RF filters and power distribution networks. Furthermore, we quantify the uncertainty of our predictions in the extrapolated band using Bayesian inference and approximate it using variational techniques. Along with providing the mean output prediction for out-of-band frequency values, we provide the output variance as a measure of uncertainty as well. (2) We propose Transposed Convolutional Networks to model spatial correlations in the design space. The design space comprises of all the geometrical and material parameters characterizing the response. The convolutional networks can extrapolate the design space in as high as 11 dimensions because of inducing spatial bias into the model. The technique is applied to 5G band RF filters as well as power delivery applications. Furthermore, we also focus on inverse design of electronic systems. The goal in inverse design is to estimate the best set of design space values that generate the response space. We employ invertible neural networks to model the non-linear mapping between the design space and the response space. Using invertible networks, we can find the exact posterior distribution of a high-dimensional mapping design space for a given desired target. The technique has been applied in the areas of high-speed signaling and power delivery. We

use neural network discriminator in a adversarial manner to come up with multiple inverse design solutions.

CHAPTER 1
INTRODUCTION

1.1 Motivation

With the tremendous growth of the semiconductor industry, compute power and memory have become cheap and accessible. One interesting outcome of this growth has been the adoption of Machine Learning (ML) into several fields traditionally dominated by physics and mathematics [1, 2, 3, 4, 5, 6, 7, 8, 9]. Solving electrically large systems in analysing their electromagnetic, thermal or mechanical behavior can be a time and memory intensive process. But, as is well known today, such analyses become inevitable with (a) the increase in operating frequencies, (b) the scaling in system and device size, and (c) the hybrid nature of different components packaged in close proximity. As system complexity increases, design cycles become longer as each design iteration requires multi-variable analysis of electromagnetic structures. Contemporary examples of such complexity are mmWave systems where multiple chiplets and micro-wave components are integrated on a single substrate or package [10][11].

Finding an optimal design in such large and complex design spaces is one element of the design cycle. But, it is also essential to know the "what-ifs" of fabrication process and manufacturing variations that are known to impact the final product. The need to estimate the response of a design in the presence of these known and unknown variations makes Design Space Exploration (DSE) an indispensable step in the design cycle. A typical flow chart for DSE is shown in Figure 1.1. DSE is an extensive analysis process that can be described in five steps. The first step is to identify a set of microwave component or system design parameters $\{x_1, x_2, ...x_N\}$. The next step is to identify the design space bounds $\{[a_1, b_1], [a_2, b_2], ...[a_N, b_N]\}$ for each parameter x_i. This is followed by selecting a

set of strategically identified designs for simulations or analysis in the frequency range of interest $[f_1, f_M]$. Using the simulation/analysis results, an electrical model is developed for performing DSE. This model can now be used to estimate the electromagnetic response of the microwave component or system for (1) Identifying worst case response based on input parameter combinations (2) Finding an optimal solution for a weighted aggregated cost function or a pareto optimal solution for multiple objectives and (3) Analyzing sensitivity for identifying the most critical parameters that affect the system response.

1.2 Summary of Contributions

More specifically, contributions of thesis can be summarized as:

- We develop specialized Long Short Term Memory Recurrent Neural Networks for frequency response extrapolation. We then introduce Hilbert transform to correlate real and imaginary part of the signal to enable causal complex-valued extrapolation. Harnessing the Variational Inference based Bayesian approach, we assess the uncertainty of predictions in the extrapolated space. In short, we propose a probabilistic machine learning framework for extrapolating the frequency response of distributed physical circuits. For the structures where there is hidden dependency between higher and lower frequency features, we propose a method to extrapolate the response while providing confidence intervals harnessing Bayesian recurrent neural networks (RNN) thereby avoiding extensive simulations and saving computational time. To address complex-valued impedance, Hilbert Transform is used to relate the real and imaginary parts where a Hilbert based RNN architecture is proposed called Hilbert Net to extrapolate the frequency response. We apply the technique to four applications (1) a simple microstrip transmission line circuit for proof of concept (2) Coupled waveguide filter operating in D-band comparing with measured results, (3) 5th order interdigital bandpass filter for 28GHz band and (4) complex stack-up power delivery network having a sharply changing response to test the framework limits. Results

show that our architecture performs accurate extrapolation with a normalized mean square error of 0.008 ohms squared with 95% confidence for a typical power delivery network. Using probabilistic networks, we achieve a tight confidence bound on our results. Furthermore, the reliability of Hilbert Net is assessed as to how far the response can be extrapolated.

- We propose a machine learning framework for predicting frequency response of a power delivery network as a function of its extrapolated multidimensional geometrical and material parameters. The proposed approach comprises of an ensemble of architectures: (1) Fully Connected Upsampler for latent code generation (2) Convolutional Decoder to learn the frequency response from the latent code. The 14D design space is converted to a Lth dimensional code which entails the frequency response information. With the proposed architecture, a root mean squared error of 0.004 ohms is achieved when compared to the true value. We focus on extrapolation of design space parameters while training on in-band values. We also illustrate how frequency poles move with varying design space exploiting parameter sensitivity in different frequency bands.

- We achieve an inverse mapping of a power delivery network's physical and geometrical properties to the impedance specification over a wide range of frequency through invertible neural networks. Training the machine learning network involves learning over a variety of stackup specifications. Once the invertible network is trained, the user can specify target impedance spec and obtain the probability density of the values of the design space that most likely satisfies the design specifications.

- We present a machine learning based tool to quantify uncertainty for prediction problems regarding signal integrity. Harnessing invertible neural networks, we convert the inverse posterior distribution given by the network to address uncertainty in frequency responses as a function of design space parameters. As an example, we con-

sider a differential plated-through-hole via in package core and predict S-parameters from its geometrical properties. Results show 3.3% normalized mean squared error when compared with responses from a fullwave EM simulator.

- We present Adversarial Invertible Neural Networks - a machine learning based inverse design technique that generates the most suitable design solution for a desired response for microwave and electronic system applications. We harness flow-based invertible neural networks that are trained adversarially to make possible the inverse design of high-dimensional parameterized structures. We illustrate our technique on 3 examples: (1) a simple patch antenna, (2) Substrate Integrated RF waveguide and (3) a differential via pair in package. We compare our approach with other state-of-the-art inverse design techniques as well. Results show that not only does the proposed approach provide the most suitable inverse design solution in comparison to its competitors but also suggests new inverse solutions to aid the designer. The posterior generated by the model depicts the uncertainty associated with the solution.

1.3 Outline of the Dissertation

The rest of this thesis is organized as follows: Chapter 2 provides a brief background on design space exploration strategies, machine learning preliminaries, neural networks, Gaussian Processes and Bayesian uncertainty quantification (UQ), Chapter 3 presents a LSTM-RNN network for frequency response extrapolation for distributed electromagnetic structures, Chapter 4 presents a transposed convolutional neural network architecture to predict high-dimensional frequency responses from design space of a EM structure, Chapter 5 presents the inverse design techniques to predict the design space from a desired response space with the application of a power delivery network, Chapter 6 developes techniques for quantifying uncertainty using INNs for a differential via pair and also compares various architecture for inverse design for a high-speed channel link, Chapter 7 discusses a novel adversarial invertible neural network for high-dimensional inverse design, followed

by summary and future work in Chapter 8.

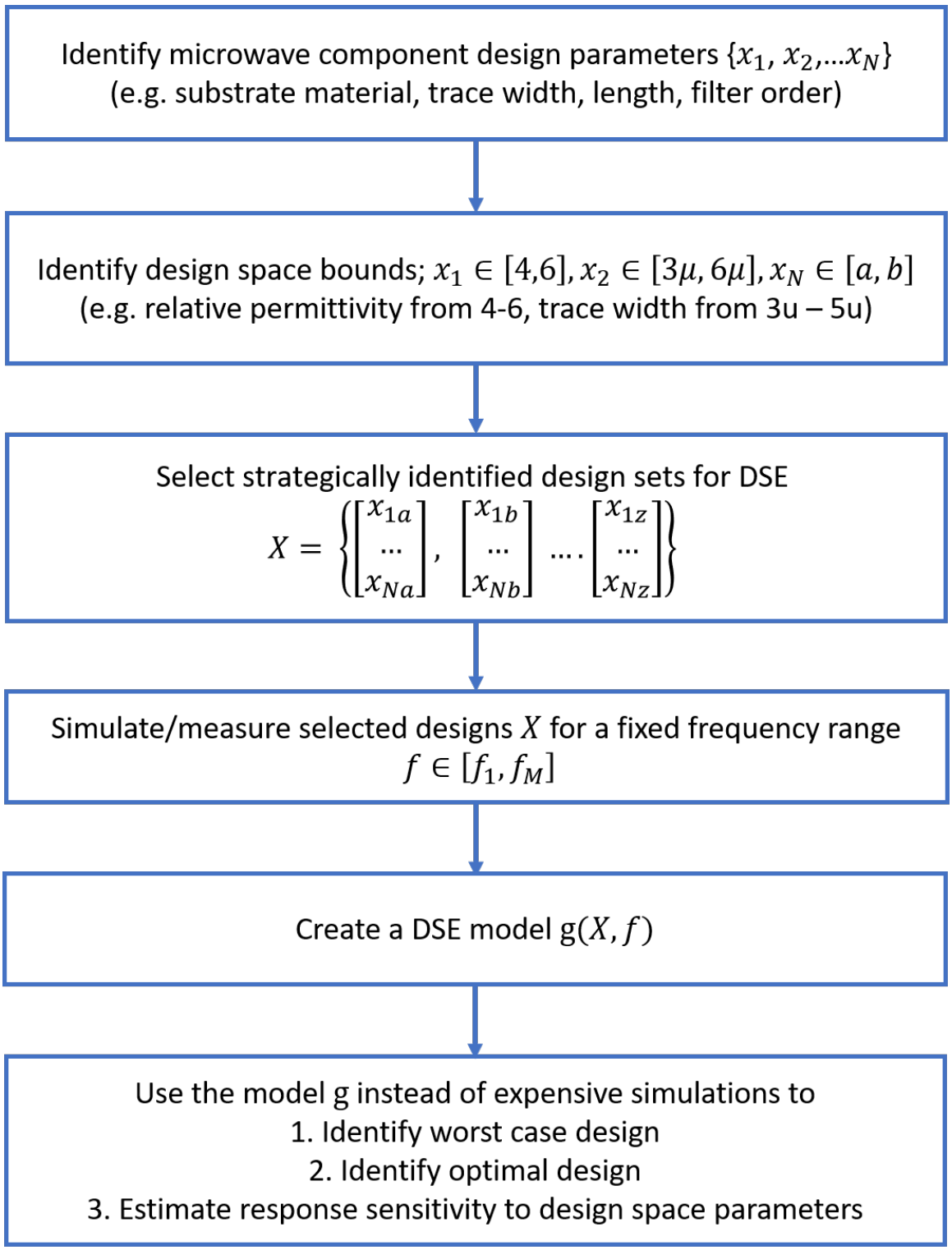


Figure 1.1: Design Space Exploration

CHAPTER 2

LITERATURE SURVEY

2.1 Design Space Exploration methodologies

DSE algorithms have been around since the early 1960s. The initial DSE techniques involved statistically derived Design of Experiments (DoE)[12] based analysis. An important element in the design of mmWave systems is the estimation of the electromagnetic response of several sub-components making up the system. For DoE like methods, based on [13], the number of measurements (or full-wave EM simulations) typically scales exponentially with design space size which becomes a bottleneck, especially in high dimensional scenarios. For quantifying uncertainty for DoE based approaches, stochastic DSE and optimization methods have been developed [14], [15], [16]. Though known to work for linear or weakly non-linear spaces, these methods generally do not scale well with increasing dimensionality and non-linearity of design spaces. To address these limitations, evolutionary algorithms for DSE were developed in the early 2000s. The evolutionary algorithms were based on biological mechanisms such as reproduction, survival, natural selection or swarm intelligence and communication. Unlike previous methods, evolutionary techniques made no assumptions about the nature of the design space. Some of these methods applied to electrical system design, include Particle Swarm Optimization (PSO) [17] and Genetic Algorithm (GA) [18]. However, a few drawbacks of such evolutionary algorithm based methods were (a) they required some heuristics (such as population size, mutation factors, accelerate constant, inertia weight) which played an important role in performance and efficiency of the algorithm, but could only be identified by screening or experience as they have no known dependence on characteristics of the design space [19], (b) a large number of function evaluations were required and (c) convergence rates were problem dependent, often leading

to long compute times. ML based DSE approaches overcome several of these challenges ; namely, (a) they can be applied to non-convex response surfaces covering a much larger design spaces (b) provide techniques for quantifying uncertainty around predictions and (c) have implicit sensitivity information built into the algorithms.

DSE methods were developed to explore the system response within a bounded predetermined design space. But what-if the design space changed a little? Or the behavior at certain frequencies beyond those analyzed was required?

Most of the above mentioned DSE approaches have one common drawback, namely the inability to extrapolate accurately, the system response outside the predetermined design space. Hence, there are two different set of parameters that are of key interest to the designer. One is the design space parameters that have been discussed so far, the other set of parameters are the frequency points or the frequency sweep over which the system response is evaluated. Extrapolation becomes useful for several reasons that include the following: i) unknown resonances may reside just outside the bandwidth of interest which may affect design decisions; ii) the parametric space is high-dimensional making it expensive to rely on expensive 3D simulations to extract the response; iii) the extrapolated space might provide insight into alternate design solutions and iv) the information obtained might point towards additional simulations to learn more about the component. In all these scenarios if data can be obtained in the extrapolated space through additional simulations, measurements or other means, that can only help towards developing a more robust model. The main reason for the poor extrapolation capability of conventional DSE approaches is the lack of ”memory” and ”learning” in the system. We focus this article on this aspect of the problem, namely extrapolation along the frequency axis and in the design space for RF and microwave component designs. By adding this memory and learning into the system, models that can look beyond the bounds and help answer the what-ifs outside the space can be created with reasonable accuracy, without having to add additional data points from the extended space. Moreover, these models, along with the prediction of the system re-

sponse, in the extrapolated region, also provide uncertainty quantification that would aid the designer to understand if more data samples are required in the extrapolated space.

One aspect of DSE is also the frequencies for which the designs are simulated. Higher solution frequencies lead to finer meshes and more expensive system forward solves. This limits the number of frequency points that can be simulated and makes it important to have accurate interpolation and extrapolation techniques to estimate the frequency response outside the solved points.

Extrapolation in design space has been discussed using evolutionary algorithms [20], but these have not been broadly deployed for DSE methods in RF designs. For frequency extrapolation, however, there are a few documented methods available in the literature. The Cauchy Method [21] is one example, wherein the frequency response is modeled as the ratio of two polynomials. Given sampled data, the coefficients of the polynomials can be determined using singular value decomposition (SVD) techniques implemented using total least squares (TLS) solution. The method then relies on the theory of analytical continuation to extrapolate the response in frequency. This method has been applied to smooth frequency responses for extrapolation but tends to pose problems for non-smooth responses. Another method to extrapolate in frequency has been presented in [22], where the frequency response is represented as a sum of orthogonal polynomials. A GA is then used to extract the necessary extrapolation parameters. It uses a fixed analytical function describing the response without the ability to generalize on multiple frequency responses. Another possibility is to rely on the frequency samples being correlated where this information is used for extrapolation. This translates to constructing a generative model for predicting the next sample given the history of samples. Under these constraints, series forecasting techniques can be applied. Historically, autoregressive integrated moving average (ARIMA) models have been used for predicting the next value in stationary signals [23]. Such a technique relies on finding seasonal trends and local periodicities in the signal. One common pitfall of this method is that it finds fewer parameters to describe the

signal, hence the extrapolation may suffer in accuracy. Furthermore, authors in [24] pose the extrapolation problem as an optimization challenge and solve it using neural networks which makes the extrapolation model smooth outside the training region. In this article, we provide an outline of two ML based architectures that accomplish extrapolation in both design space and frequency with high accuracy, for microwave and RF components with over ten design variables and frequencies beyond 150GHz.

2.2 Machine Learning Preliminaries

ML is an alternative technique for creating mappings between multiple inputs and outputs for a system. ML methods are often looked at as "black box" approaches where mappings are generated without any domain expertise. However, for addressing electromagnetic problems, using domain expertise, such as the shape of the frequency response, the range of the physical and geometrical parameters of the structure and the resonant frequencies of the system become important. A major advantage of a fully trained ML model is that it can predict the output response in far less computational time than traditional methods, as illustrated in subsequent sections. A perceptron, shown in Fig. Figure 2.1a is the backbone of any ML architecture. It computes an output value that is a non-linear function of a weighted sum of the inputs. A typical Neural Network (NN) model is a layered network of many such interconnected perceptrons. The NN consists of an input layer responsible for data pre-processing, hidden layer(s) that perform computations on the transformed input and an output layer for post-processing, as illustrated in Fig. Figure 2.1b. The NN thus forms a mapping between a set of inputs x and outputs y. The "learning" in the network is performed by back-propagating the loss function from the output layer to the input and adjusting the "weights", W, connecting the various perceptrons and the "bias", b, at each perceptron, to minimize this loss function. The loss function is a measure of similarity between the predicted output y and the true output $\hat{y}$. A gradient descent algorithm is used to find the minima of the loss function in the model parameter space $\{W, b\}$. Such a net-

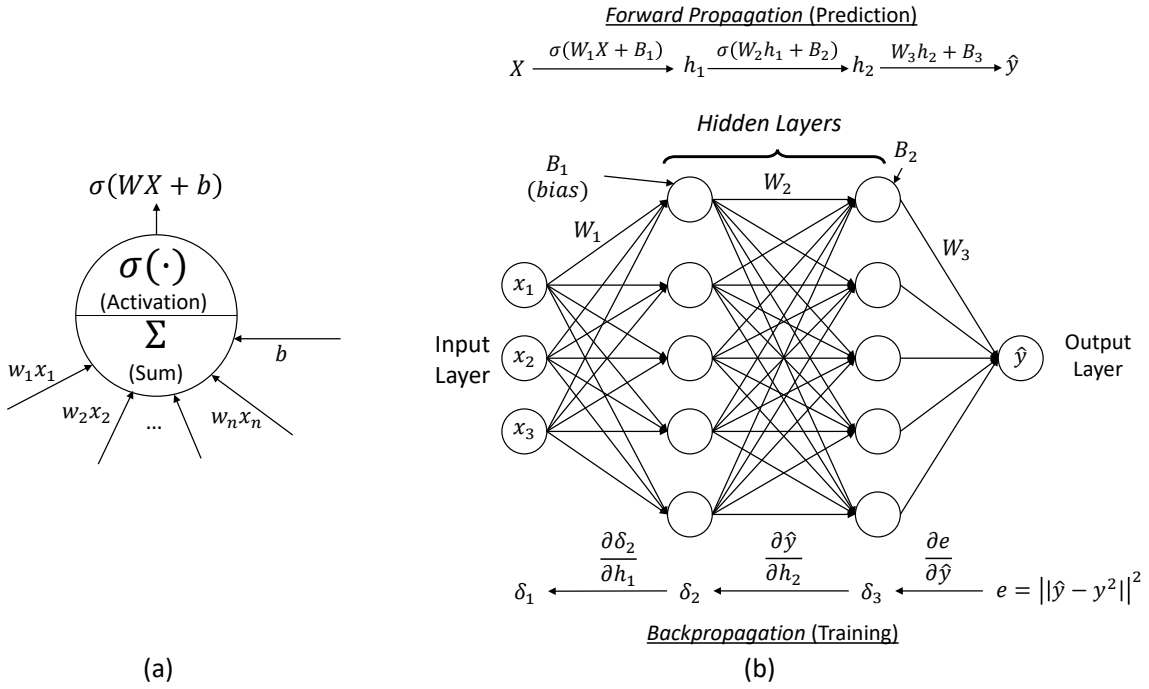
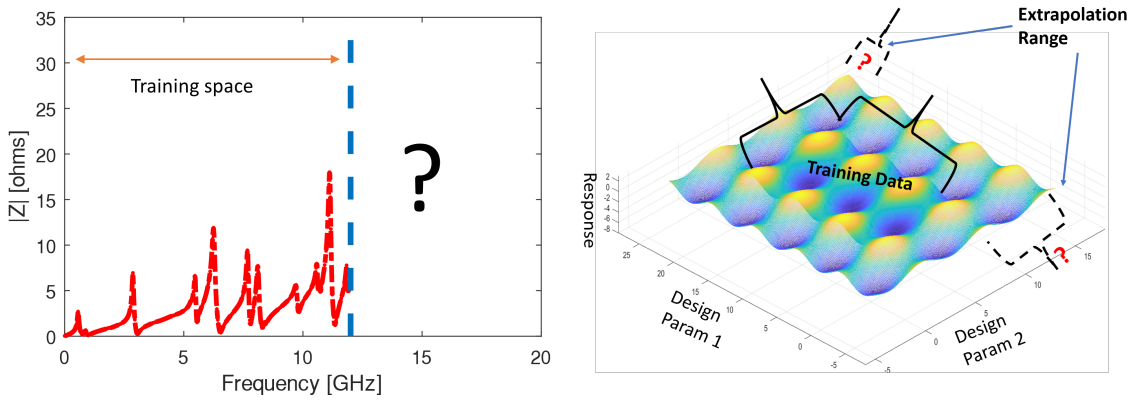


Figure 2.1: (a) Single neuron. (b) Feed forward neural network [9]

work is capable of learning but incapable of remembering the trend of inputs in time or frequency. To enable the model to be able to learn data dependencies over time, the perceptron has to be a function of not only the current input but the previous input trends and outputs. By making this change in the perceptron, the model becomes capable of predicting the future trends and hence can be used for ”extrapolation”. ML architectures formed of such compute and memory capable units are described in detail in the next section.

Design Space Exploration using ML of many core systems has been explored in [25] wherein a local search and meta-search are used. The ML architecture is contained in the meta-search which evaluates the design space and suggests potential starting states to explore. In [26], predictive models are constructed using feed-forward neural networks that are applied to hybrid main-memory design. Linear regression techniques have been developed in [27] to show that performance of a system can be accurately predicted by using a small fraction of the overall design space and by leveraging information from previously simulated data. More recently, sensitivity analysis of design parameters have been developed [28] for high-dimensional microwave problems. State-of-the-art ML methods based



(a) (b)

Figure 2.2: Illustration of (a) Spectrum Extrapolation and (b) Design Space Extrapolation in $\mathcal{R}^2$

on non-linear regression architectures have been applied to multidimensional inputs and multiple outputs in [29]. All of these ML approaches work for DSE, however, they may not be directly applicable for extrapolation of frequency spectrum or design space.

An example of a simulated circuit frequency response from DC to 12GHz is shown in Fig. Figure 2.2a while system response surface at a single frequency, for two design variables is shown in Fig. Figure 2.2b. Given these two typical examples in RF design, the objective of this article is two fold namely, 1) extrapolation of the frequency response beyond 12GHz, and 2) extrapolation of the design space, with both being enabled without adding any additional data samples from the extrapolation range.

The technical approach for achieving such an extrapolation is illustrated in Fig. Figure 2.3. As shown in the figure, based on the design space parameters $\{x_1, x_2, ...x_N\}$, selected designs, from within the design space bounds are simulated for a set of frequencies $\{f_1, f_2, ...f_M\}$ using full-wave 3D EM solvers or circuit simulators that accurately capture the system/component response h. These design parameters, and the corresponding responses, are used to create a surrogate model g, that captures the system/component response h, for generating training data within the design space. This data, is utilized to train a Design Space Extrapolator Neural Network (DSENN) $d(x, f)$ and Frequency Ex-

trapolator Neural Network (FENN), $l(f)$ as shown in Fig. Figure 2.3. Both d and l are trained to model the system response h. The training of these two networks is performed separately. The DSENN is trained using data from within the design space to predict the frequency response for a set of design parameters outside the design space. The FENN is trained to predict the response for frequencies outside the simulated spectrum by learning from the response within the spectrum. After training, given a set of design parameters that lie outside the training range as inputs, component or system response is predicted using the trained models d with l as a sub-block.

2.3 Bayesian update Rule

Bayes theorem is based on subjective probability as opposed to objective probability practiced by frequentists. In mathematical form it can be written as:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{2.1}$$

where H is the hypothesis, E the evidence, $P(H|E)$ is the conditional probability of the hypothesis when the evidence is considered (posterior), $P(E|H)$ is the conditional probability of the evidence given the hypothesis is true (likelihood), $P(H)$ is the probability of the hypothesis before the evidence is considered (prior), and $P(E)$ is the probability of the evidence under any circumstance (marginal probability) given by:

$$P(E) = P(E|H)P(H) + P(E|H')P(H') \tag{2.2}$$

where $P(H') = 1-P(H)$ is the probability of the hypothesis not being true and $P(E|H')$ is the conditional probability of the evidence when the hypothesis is untrue. The denominator term in (Equation 2.1) is generally a normalizer to ensure that the result in (Equation 2.1) is always a probability that is bounded between 0 and 1.

As an example, in Fig. Figure 2.4, the prior is assumed to be a gaussian distribution

with large variance, where the random variable is the parameter x. By making use of the likelihood and the prior, the posterior is computed with less variance. A smaller variance translates into a better confidence in the prediction, or in other words smaller uncertainty. Being subjective Bayes theorem allows for guesses where the prior distribution can be assumed, making it powerful and applicable in several areas.

2.4 Stochastic Models

Traditionally, the outputs of a NN are deterministic. The challenge with deterministic NNs is that the prediction is exact without any estimate of the probable error around predictions. This often gives an incomplete picture of the system response and could be dangerous unless the prediction uncertainties can be included. It is therefore important to quantify the uncertainty around prediction which can be achieved using Bayes by Backpropagation (BBB) [31] algorithm instead of the usual error backpropagation for training the NNs. In such a NN, instead of assuming the weights and biases in the system to be deterministic variables, they are assumed to be samples from prior distributions. The output y of the network is then a probability distribution that is computed using the given independent data input x. To obtain the distribution of outputs, the probability over all the various weights is marginalized, given the inputs:

$$p(y|x) = \int p(y|x,\theta)p(\theta|x)d\theta \tag{2.3}$$

where θ represents the set of the parameters of the weights W and bias b distribution of the model. The second term in Equation 3.13 is found using the Bayes' rule:

$$p(\theta|x) = \frac{p(x|\theta) * p(\theta)}{p(x)} \tag{2.4}$$

Hence, the output of such a NN is now a distribution computed using marginalization of the probability distributions of the weights and biases, given the input. As the output is a

distribution, its mean and variance are known. The mean is represented as the predicted output and the variance is used to calculate the upper and lower confidence bounds around the predicted mean thus, giving the end-user a quantification of uncertainty in the prediction or the error bar around the prediction.

2.5 Gaussian Processes for Machine Learning

GP is the extension of standard multivariate Gaussian distribution to infinitely many variables, where any finite number of samples form a joint Gaussian distribution [32]. The prior of GP is defined by two quantities, namely a mean μ and a covariance matrix K, given by:

$$y = f(x) \sim \mathcal{N}(\mu(X), K_X) \tag{2.5}$$

where N represents a GP. From (Equation 2.5) the mapping between the input and output is enabled through the GP. For general non-linear regression, a constant mean function $\mu(x) = m$ is used [33]. The kernel function $K(\boldsymbol{x})$ that describes the relation between points in the function is written as:

$$K(\boldsymbol{x}) = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_t) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_t, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_t, \boldsymbol{x}_t) \end{bmatrix} \tag{2.6}$$

Appropriate kernel functions can be applied to capture different patterns in the dataset. For example Matern kernels can be used when the function is less smooth. A commonly used kernel is the automatic relevance determination (ARD)[33] Matern 5/2 function given

by[34]:

$$k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \sigma_f^2 \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) e^{-\sqrt{5}r}$$
$$r = \left(\sum_{d=1}^{D} \frac{\left(\boldsymbol{x}_{i,d} - \boldsymbol{x}_{j,d}\right)^2}{\sigma_d^2}\right)^{\frac{1}{2}} \tag{2.7}$$

where σ_f and σ_d are the hyperparameters of $K(\boldsymbol{x})$. These hyperparameters are updated during the training process by minimizing the negative log marginal likelihood of the GP to improve learning.

We can also combine different standalone kernels to construct new kernels, which can then be applied to capture more complicated function behaviors, as illustrated in Fig. Figure 2.5. Further details on kernels are provided in [35].

Once the GP model is trained using the dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{Y}\}$, it can be used to predict the unknown response y^* for a new set of input data $x^* \in \mathbb{R}^{M \times D}$ using the relationship:

$$p(y^*, Y|x^*, X, \theta) = \mathcal{N}\left(\begin{bmatrix}\mu X \\ \mu x^*\end{bmatrix}, \begin{bmatrix} K_X & K_{X,x^*} \\ K_{X,x^*}^T & K_{x^*,x^*}\end{bmatrix}\right) \tag{2.8}$$

where θ is the set of hyperparameters used as part of the training process.

During the training process, our goal is to find the best hyperparameters, θ, that fits the data and model. Fixing θ can create a combination of data and parameter related uncertainties due to any inaccuracies related to the model used for prediction. Therefore, we integrate over all possible θ and use a weighted sum of confidence intervals where the bounds obtained with multiple θ values affect the final confidence bound. This can be written as [9]:

$$p(y^*|x^*, D_t) = \int p(y^*|x^*, D_t, \theta) p(\theta|D_t) d\theta \tag{2.9}$$

where $D_t = (X_t, Y_t)$ is the data at the tth iteration of the training process. At a test point x^*, the model predicts a distribution, $p(y^*|x^*, D_t)$, that no longer depends on θ

and is a weighted sum of all possible distributions corresponding to fixed hyperparameter $p(y^*|x^*, D_t, \theta)$. We can use Markov chain Monte Carlo (MCMC) [36] to learn the analytically intractable distribution $p(\theta|D_t)$. Once the GP is trained, the predictions and confidence intervals that also accounts for parameter-related uncertainties can be obtained as in [37]. A

2.6 Normalizing Flows

In probability theory, we can use the change of variables formula for finding the pdf of a transformed variable using a independent variable given as:

$$p(y = f(x)) = p(x) * |J_{yx}|^{-1} \tag{2.10}$$

where $J_{yx} = \nabla_x(f)$ and f can be modeled as an invertible block from previous discussion. From [38], the name "normalizing flow" can be interpreted as the following: (1) "Normalizing" means that the change of variables gives a normalized density after applying an invertible transformation and (2) "Flow" means that the invertible transformations can be composed with each other to create more complex invertible transformations.

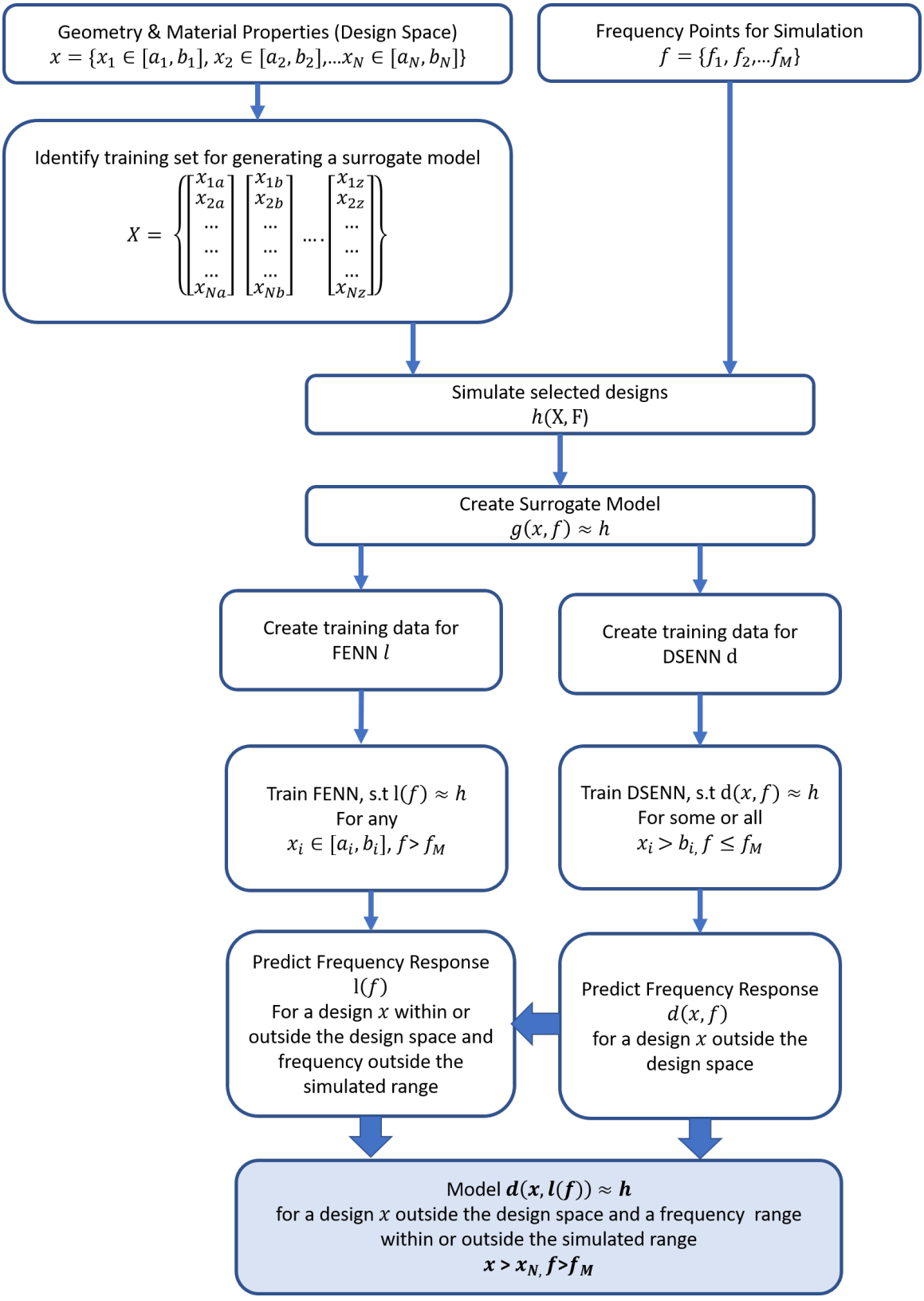


Figure 2.3: Design space and spectrum extrapolation using Frequency Extrapolator Neural Network (FENN) and Design Space Extrapolator Neural Network (DSENN)

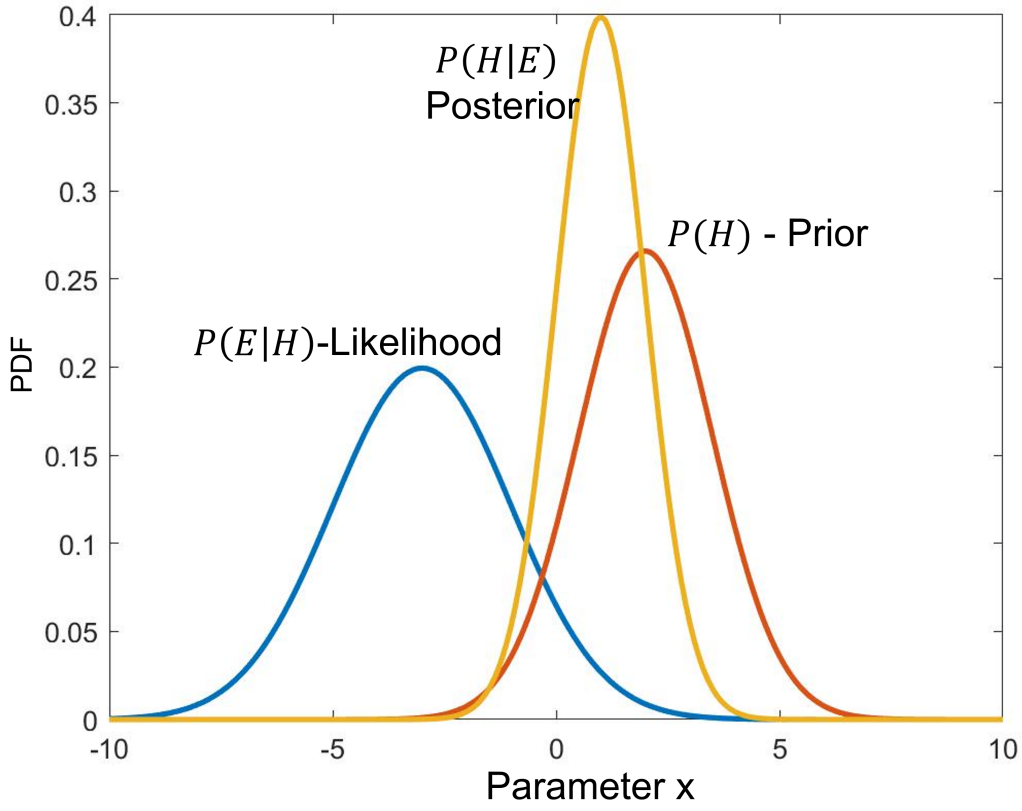


Figure 2.4: Updating prior to obtain the posterior after seeing the likelihood according to (Equation 2.1). Here, prior $P(H)$ is assumed to be a Gaussian distribution with large variance. After observing the evidence based on the hypothesis, the prior is updated to the posterior $P(H|E)$ with less variance, and hence with more confidence [30]. Note: This is shown for illustration purposes only since the distributions can be arbitrary.

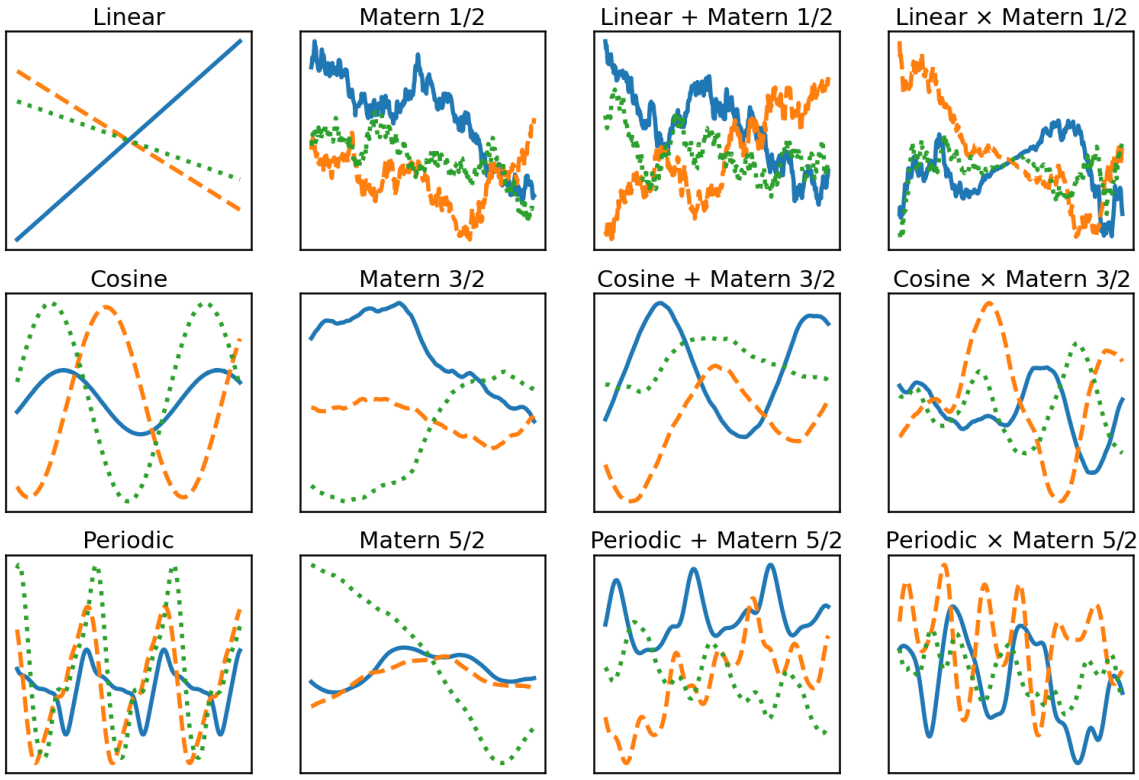


Figure 2.5: Different GP kernels and their combinations

CHAPTER 3
NEURAL NETWORK ARCHITECTURE FOR FREQUENCY RESPONSE EXTRAPOLATION

In this section, we present the techniques and methodologies for frequency response extrapolation.

3.1 Problem Statement

Distributed electromagnetic structures, characterized by their frequency responses, have important information embedded in them about the operation of a circuit. In most scenarios, only band-limited data is available because broadband data might be computationally expensive to simulate or unavailable. To this end, extrapolation in the frequency domain is a compelling concept because it enables us to estimate the frequency response over a larger range without performing extra measurements or simulations. Formally, the problem of frequency extrapolation is to estimate the out-of-band frequency response given in-band frequency response. Mathematically, considering a N-point in-band frequency response data: $D = \{f_i, Z(f_i)\}_{i=1}^{N}$, the goal is to find the extrapolation frequency response $Z(f_j)$ where $j = N+1, ..., N+F$ for F points in the extrapolated band. An example architecture of the extrapolation NN is shown in Figure 3.1. The architecture consists of three main blocks, a fully connected up-sampler, the DSENN and the FENN, connected in that order. It is important to note that both the DSENN and FENN can be used in isolation as well. However, when used as a system as explained in Section II, the input layer is formed using the design space parameters that could be either geometry and/or material properties. The dimension of the input space is generally much smaller than the number of frequency points used to generate a system response. Therefore, the inputs are first up-sampled to a larger dimensional latent space using a fully connected up-sampler such as a feed-forward

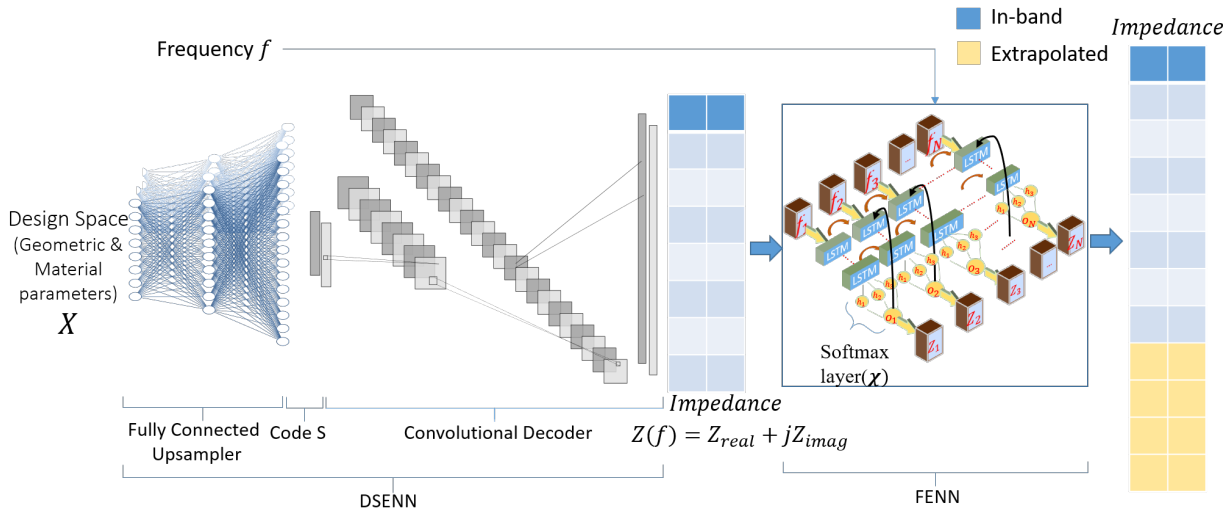


Figure 3.1: Neural network architecture for Design Space and Frequency Extrapolation

NN. This is then followed by a convolutional encoder. The outputs of the convolutional encoder are the real and imaginary components of the frequency response of the system. This frequency response is then fed to a modified Recurrent Neural Network that is trained to predict the extrapolated response from the inputs. Thus, in its entirety, the ML architecture can predict an extrapolated frequency response of a design outside the design space. The details of the convolutional encoder and modified RNN are given in this section.

3.2 Recurrent Neural Networks (RNN) for Spectrum Extrapolation

As mentioned in previous chapter, for a ML model to recognize a pattern and extrapolate it, a regular perceptron needs to be replaced with a modified perceptron capable of processing current and past inputs. The resulting network of such perceptrons is called a Recurrent Neural Network (RNN), as shown in Figure 3.2. A circuit response in frequency is treated as a set of sequenced data where f_i is the frequency at index i and Z_i is the response at index i. The hidden state at index i, h_i is a recurrent state that depends on the previous trends in the input. This unit is the backbone of the recurrent architecture. This state, h_i, captures how the response has changed over the past frequency samples and how it will behave at the next frequency sample. As h_i changes from the beginning of the range to the

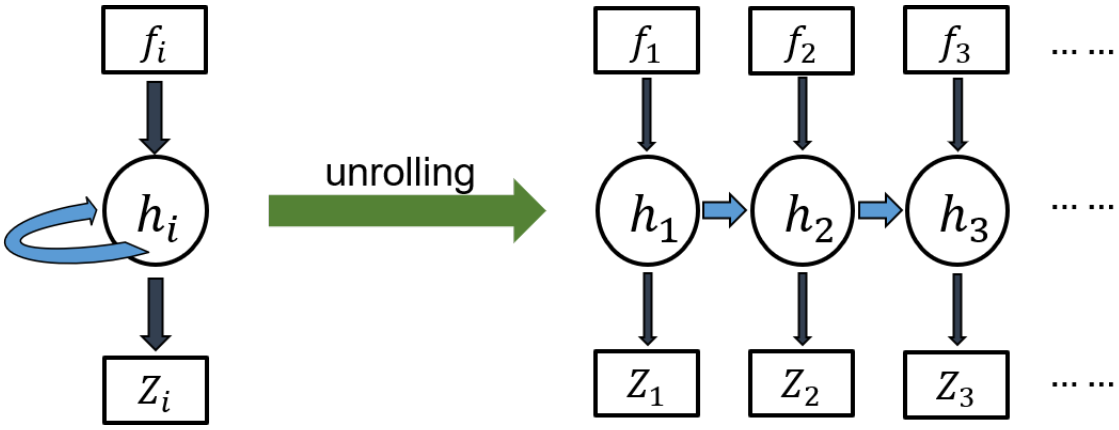


Figure 3.2: Recurrent Neural Network (RNN)

end, it holds the pattern of early frequency response trends. These kind of sequence models capture seasonal trends. The hidden state provides a mapping between the input and the output.

$$h_m = tanh(\boldsymbol{W_{hh}} h_{m-1} + \boldsymbol{W_{xh}} x_m) \tag{3.1}$$

$$y_m = \varsigma(\boldsymbol{C} + \boldsymbol{W_{hy}} h_m) \tag{3.2}$$

where x_m is the current input, h_{m-1} is the hidden state at the previous index, h_m is the current hidden state to be computed, and $\boldsymbol{W_{pq}}$ is the weight matrix of size pxq mapping it from the p layer to the q. $\varsigma(\boldsymbol{x}) = e^{x_i}/\Sigma_j e^{x_j}$ is the softmax activation function i.e. the normalized exponential function. This type of network shares the parameters that enables it to learn sequences and patterns. Moreover, the hidden state stores essential features necessary for the generation of the next value. Therefore, by adding a hidden state (can be a scalar or a vector) to the model, we can add "memory" which makes the model "recurrent". This is the main reason for using a recurrent neural network(RNN) for extrapolation can be justified. Such an architecture enables the network to learn from the past and the present, and hence to extrapolate. There are, however, several challenges in the training of a RNN. As the complexity of the problem increases, the amount of training data required and the number of recurrent layers of the network increase. Training a deeper model with more

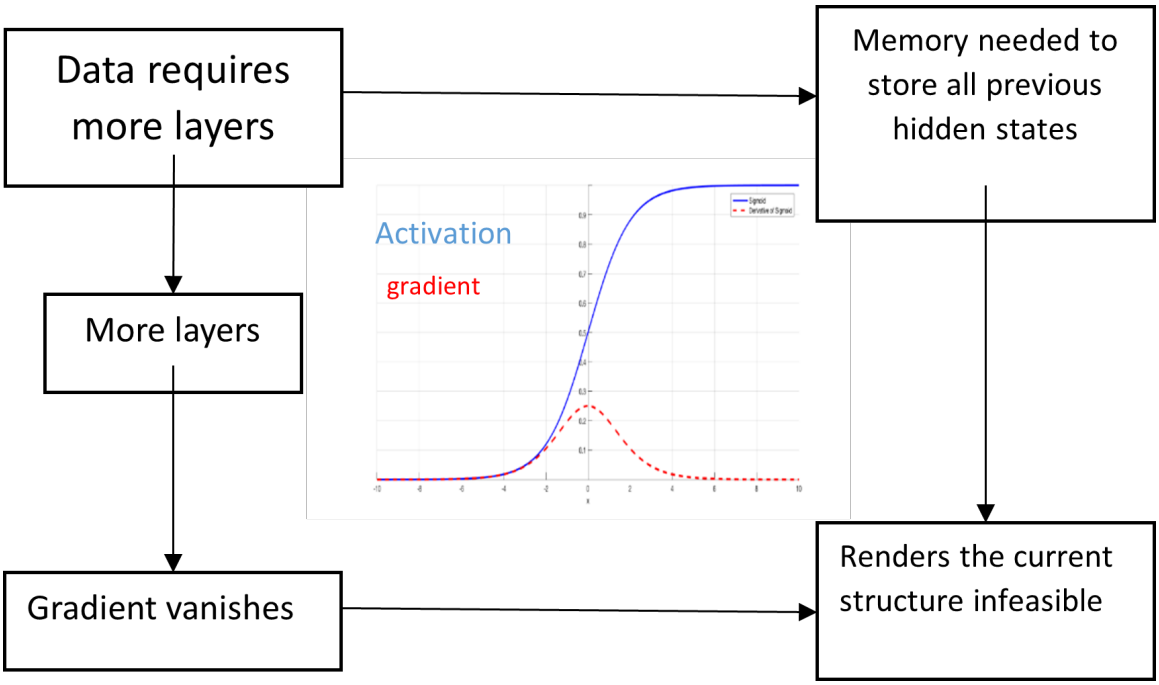


Figure 3.3: Challenges in modeling of a vanilla RNN

layers presents a challenge causes the gradient used for backpropagation to drop down to zero, the effect called gradient vanishing [39]. In addition, training for a large amount of data and tracking long sequence of input patterns requires more memory. We illustrate this effect in Figure 3.3.

To circumvent these issues, long-short-term memory networks (LSTM) can be used as described in [40]. The LSTM network is a special type of RNN with a unit cell as shown in Figure 3.4. It consists of 4 perceptron gates: (1) a forget gate f_t, (2) an input gate i_t, (3) a cell gate g_t and (4) an output gate o_t. When the hidden state h_i in the traditional architecture is replaced with a LSTM unit cell, it becomes a LSTM-RNN. The drawback of increased memory requirement is resolved with this architecture as it does not remember "all" the information from the past. It is trained to remember only what is important for extrapolation and forget everything else. These networks can have smaller number of layers compared to traditional RNNs, thus overcoming the problem of vanishing gradients as well. Mathematically,

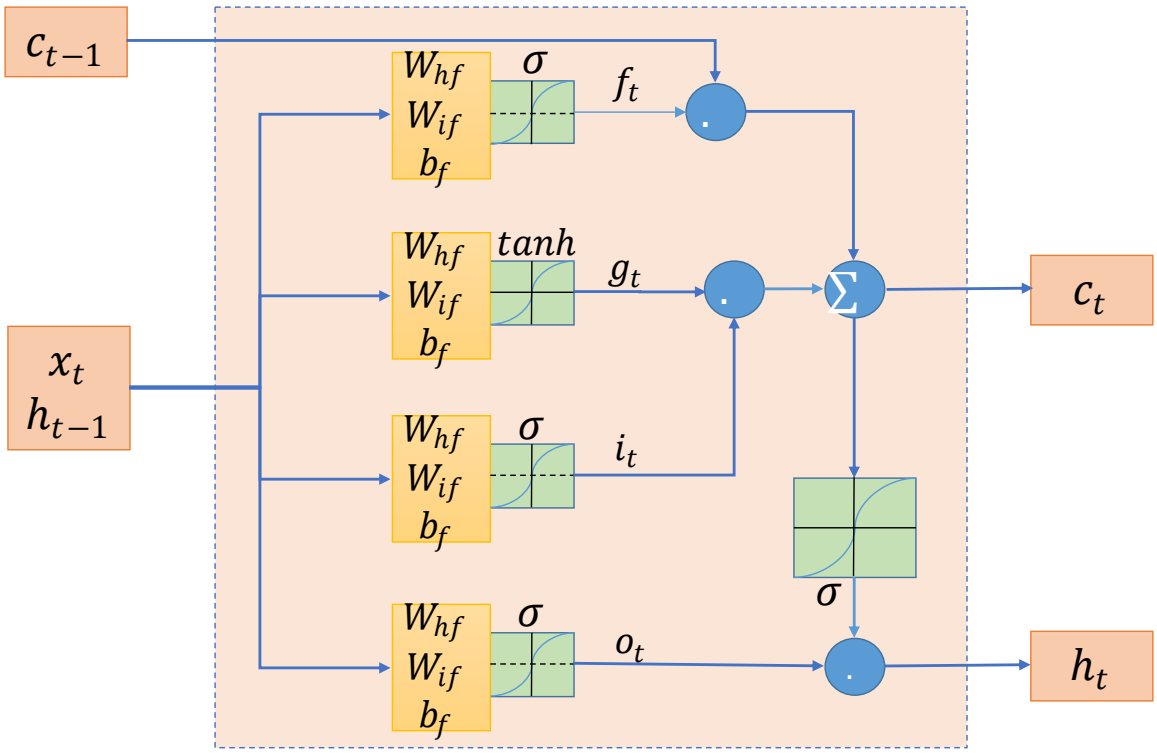


Figure 3.4: Long Short-term memory unit cell [39]

$$fr_m = \sigma(\boldsymbol{W_{fi}} x_m + \boldsymbol{W_{hf}} h_{m-1} + b_f) \tag{3.3}$$

$$i_m = \sigma(\boldsymbol{W_{ii}} x_m + \boldsymbol{W_{hi}} h_{m-1} + b_i) \tag{3.4}$$

$$g_m = tanh(\boldsymbol{W_{ig}} x_m + \boldsymbol{W_{hg}} h_{m-1} + b_g) \tag{3.5}$$

$$o_m = \sigma(\boldsymbol{W_{io}} x_m + \boldsymbol{W_{ho}} h_{m-1} + b_o) \tag{3.6}$$

where c_m is the cell state or memory and $\sigma(r) = \frac{1}{1+e^{-r}}$ is the sigmoid function that scales the input in a non-linear fashion between 0 and 1. The $\boldsymbol{W}$s and $\boldsymbol{b}$ are the model parameters. They are learnt by the network during the forward and backward passes. As can be seen from equations (3.3)-(3.6), LSTMs learn dependencies that are important enough

to retain and unnecessary enough to forget. The forget gate produces the values between 0 and 1 depending on the importance of the previous input for a given future sample. The input gate produces a value between 0 and 1 based on the current input and the previous hidden state. The cell gate is also a function of the previous hidden state and the current input state. This is important since it dictates the memory being stored. The input gates and the cell gates are multiplied and added to the forget gate to compute the cell state c_m. This enables the network to capture local periodicities and apply them for prediction farther in temporal space as compared to the training range.

3.3 Hilbert Transform for causal extrapolation

Consider a complex valued signal with redundant negative frequency components i.e. an analytical signal. In this case, we represent the impedance Z as a function of operating frequency f as:

$$Z(f) = Z_r(f) + jZ_i(f) \tag{3.7}$$

where Z_i is the reactive component and Z_r is the resistive component of the impedance. Given its Hermitian nature, we treat the impedance as an analytical signal. Given that the Hilbert transform exists and $Z(f)$ is analytic in the upper half plane of the imaginary part of Z, the Kramer-Kronig relation are satisfied. For such a signal, the imaginary part can be expressed as a Hilbert transform of the real part as:

$$Z(f) = Z_r(f) - jH(Z_r(f)) \tag{3.8}$$

where $H(\cdot)$ is the hilbert transform and is defined as the convolution of $1/\pi f$ with the input signal. It is given as:

$$H(x(f)) = \frac{1}{\pi * f} \int_{-\infty}^{\infty} \frac{x(f')}{f - f'} df' \tag{3.9}$$

Since, discrete frequency samples are used in practice, we make use of the discrete hilbert transform. It imposes the conditions that (i) the DFT of the complex analytic sequence needs to be zero for negative frequencies and (ii) for positive frequencies the spectrum of the analytic sequence needs to be two times the signal of the signal being computed on. In the discrete frequency domain, the discrete imaginary part ($Z_i[f]$) of the frequency response is the discrete Hilbert transform and the real part of the response($Z_r[f]$). Mathematically,

$$Z_i[f] = -HT[Z_r[f]] \tag{3.10}$$

As mentioned earlier, given the real part of the discrete impedance response, $Z_r[f]$, we set the DC component to be zero. Next, since the negative frequency part has to be zero, we double the positive frequencies uptil half the points to conserve total spectral energy. Mathematically, the analytical signal with K samples can be expressed in terms of its fourier transform, as in [41], given by:

$$HT(X[f]) = \begin{cases} X[0] & \text{for } m = 0 \\ 2 * X[m] & \text{for } 1 \leq m \leq \frac{K}{2} - 1 \\ X[K/2] & \text{for } m = K/2 \\ 0 & \text{for } \frac{K}{2} + 1 \leq m \leq K - 1 \end{cases} \tag{3.11}$$

where $HT[.]$ is the Hilbert transform for discrete sequences, $X[.]$ is the K-point real part of the frequency response signal. Computation of the Hilbert transform through this method is presented in algorithm algorithm 1. This implementation ensures the causality of the extrapolated signal [42]. Note that since computation of the imaginary part from the real part requires ideally infinite points, we approximate them by having maximum 1.5*N points. Thus, taking the hilbert transform induces some numerical errors. However, since we have the ground true values for the supervised machine learning algorithm proposed,

the errors are minimized to produce the correct output.

Algorithm 1: Computation of Hilbert Transform

Input: real part, x = Re(z)
Output: y = H(x)
1: Flip the input signal: $x_{flipped}(f) = x(-f)$
2: Form the double-sided signal $x_{DS}(f) = [x_{flipped}(f), x(f)]$
3: Take DFT: $X_{DS} = DFT(x_{DS})$
4: $X_{singlesided} = 2 * X_{DS}$
5: Compute Hilbert transform according to equation Equation 3.11.
=0
[1]

3.3.1 HilbertNet Architecture

We develop a network model that combines the sequential modeling capability of recurrent nets while providing correlation between the real and imaginary parts of a frequency domain signal through the Hilbert transform. Our approach is depicted in Figure 3.8. Algorithm algorithm 2 illustrates the steps involved where input to it is the complex-valued in band signal $Z_{in-band}(f)$ and output is the complex-valued extrapolated signal $Z_{out-of-band}(f)$ with a confidence interval determined by the recurrent net. In other words, let the dataset pair be $\{f_i, Z(f_i)\}_{i=1}^{N}$ where N is the total frequency samples already available and simulated. The goal is to find the impedance response for the future F samples, that is to find $Z(f_i)$ for $i = N+1, \ldots, N+F$ where F is the number of points in the extrapolated space. Consider, as an example, the impedance response of a typical power delivery network which increases in amplitude, on average with increasing frequency. Moreover, the density of resonances and anti-resonances that occur in a fixed frequency range also increase. Feeding this raw signal to the network will force the network to predict values increasing along the frequency axis which presents a training challenge. Thus, we preprocess the response into frequency windows. The frequency response $Z(f_i)$ for i = 1,2,3,. . . .,N where $f_N = f_c$ i.e. the cutoff frequency. Here, N is the total number of frequency samples we have for training. The whole response is divided into W windows with each window

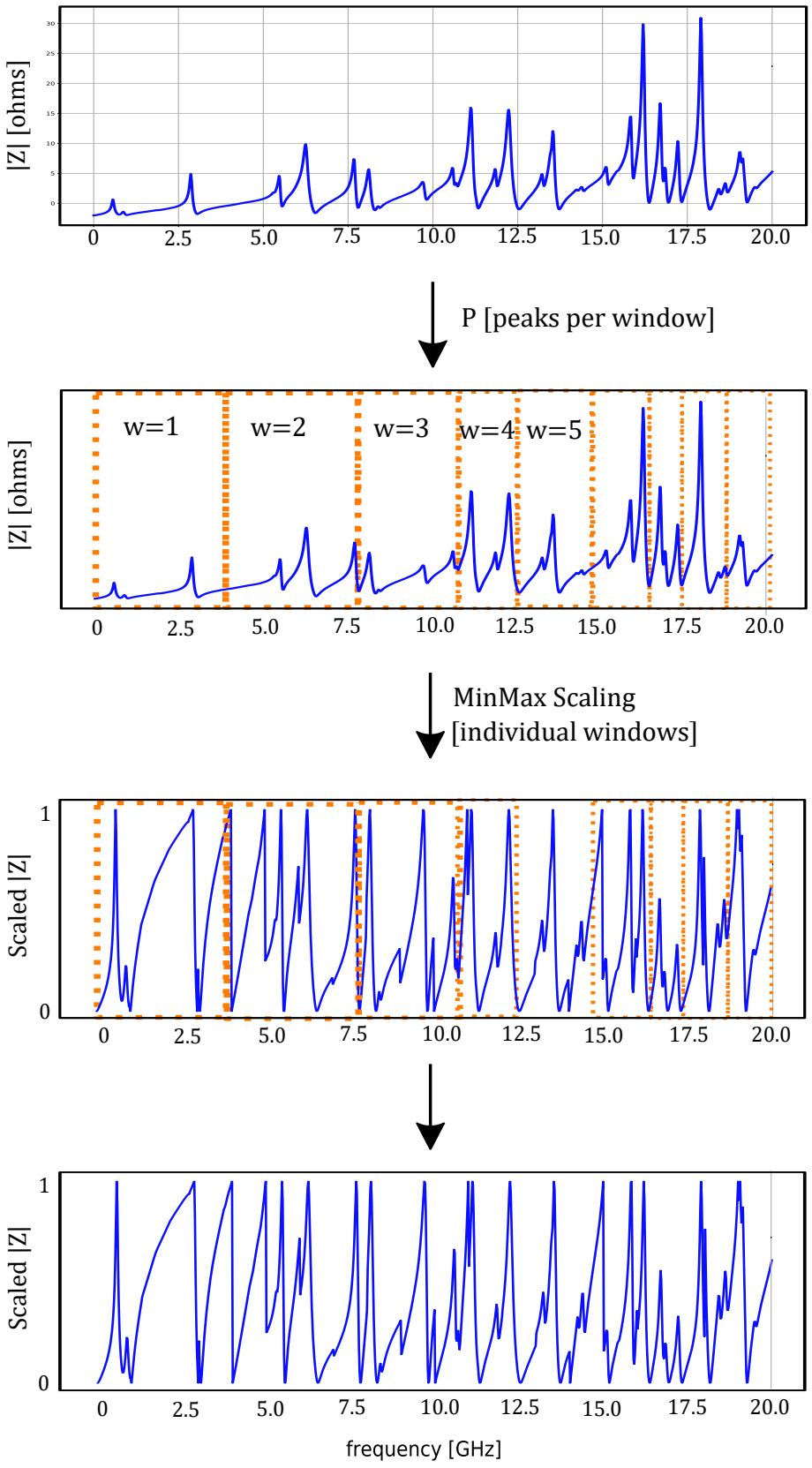


Figure 3.5: Impedance Response Scaling: window-based scaling, each window width decided by the peaks in window. Here $P = 3$.

containing P peaks or P poles/window. We can treat P as the hyperparameter for the model differing for the type of application based on the designer's needs. This means that the windows get smaller in terms of frequency samples in high frequency range. In this case, a peak is defined as a local maxima in that window but is open to other definitions for other applications. This is illustrated is Figure 3.5. For each window, the individual scaling is performed using:

$$Z_w^{'}(f_i) = \frac{Z_w(f_i) - min(Z_w(f_i))}{max(Z_w(f_i)) - min(Z_w(f_i))} \tag{3.12}$$

where $Z_w^{'}(f_i)$ is the scaled impedance response at frequency f_i for the window w and $max(.)$ and $min(.)$ are the minimum and maximum values in the current window respectively. The scaled values range from 0 to 1. This bounded input is easier for the network to learn because it produces bounded output thus making it stable. For the training range, $min(.)$ and $max(.)$ are stored to estimate the window ranges in the extrapolated region, the details of which are given in the subsequent steps. In this work, as the impedance response is complex-valued, we scale the real part of the impedance. However, this approach is applicable for the imaginary part and the magnitude as well. After performing MinMax Scaling, we form batches of sequences from the data $D = \{f_i, Z'(f_i)\}_{i=1}^{N}$. Each sequence from the frequency response has S samples. The first sequence has samples from frequency index 0 to $S-1$, the second sequence has samples from frequency index 1 to S, the third sequence has samples from frequency index 2 to $S+1$ and so on. For a total of N samples, there are $N - S + 1$ sequences. As an illustration, we show this sequencing operation for $N = 10$ and $S = 5$ in Figure 3.6.

After forming sequences, we form, feature and target sequences. Each next sequence serves as the target sequence for a feature sequence. During the training process, the feature sequence acts as a multi-dimensional input and a target sequence acts as the output sequence, thereby constructing a many-to-many recurrent network. We have depicted the

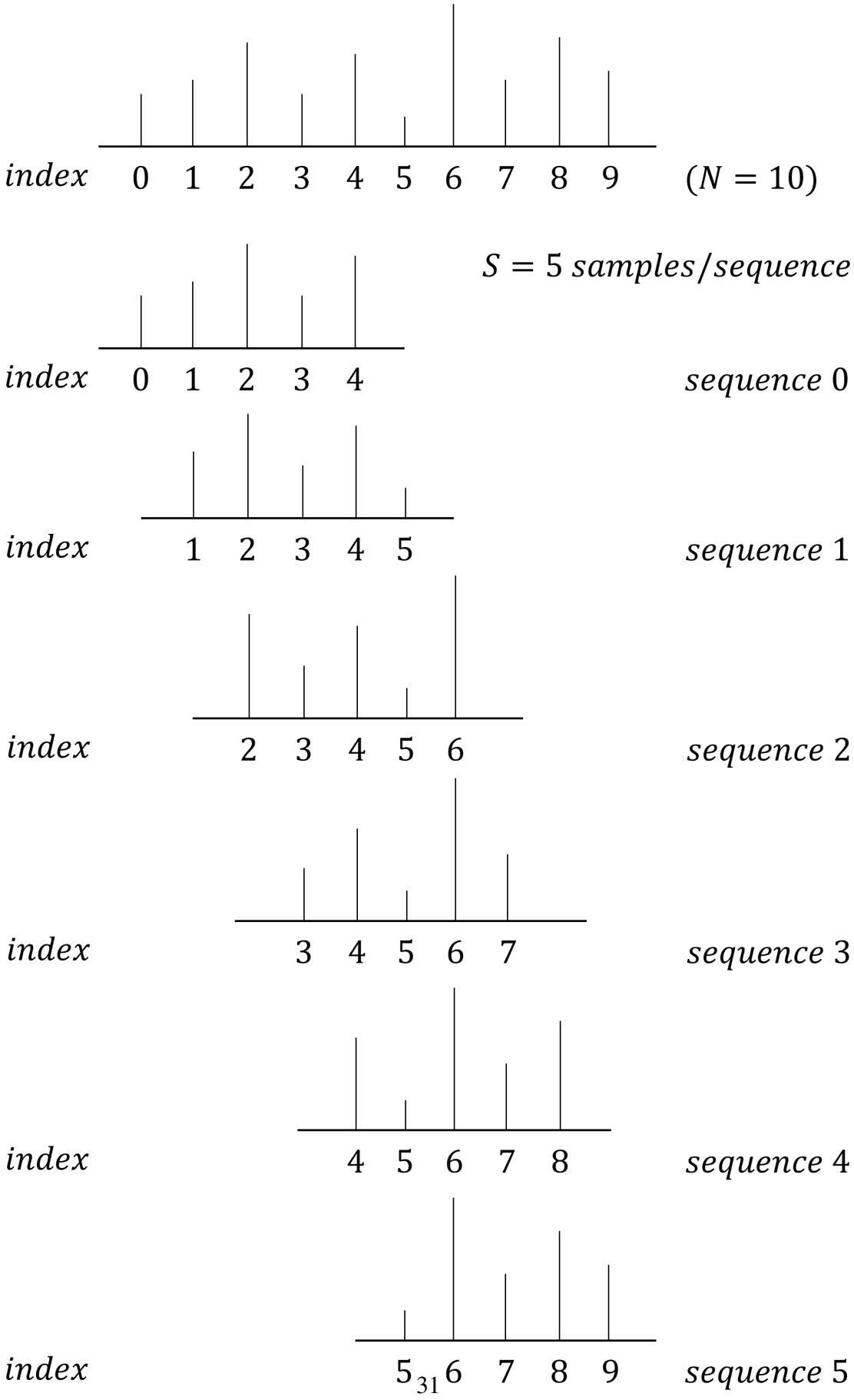
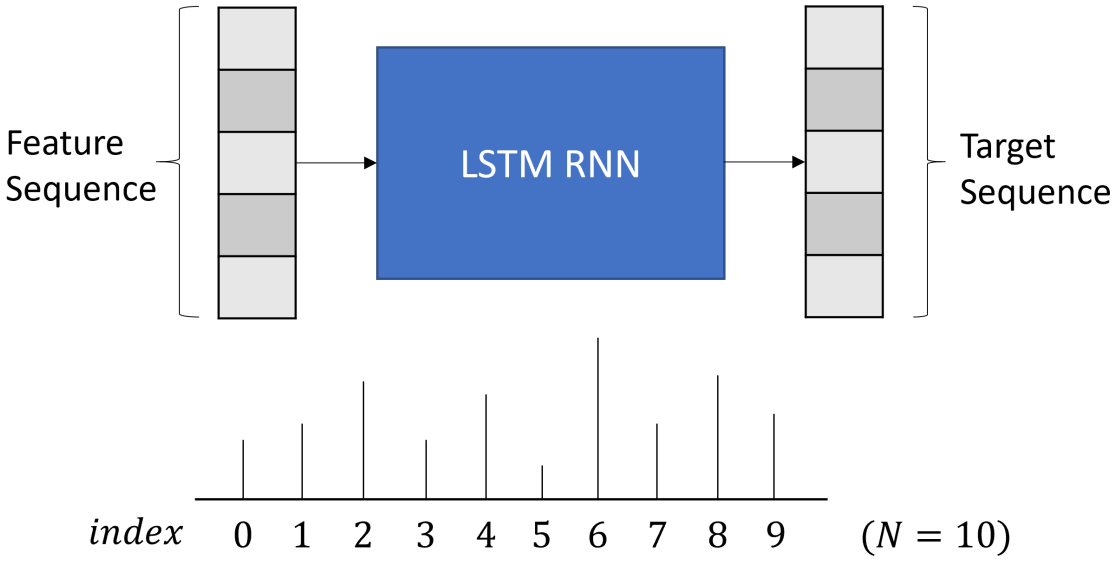


Figure 3.6: Sequencing operation for $N = 10$ and $S = 5$



Feature Sequence	Target Sequence
Sequence 0	Sequence 1
Sequence 1	Sequence 2
Sequence 2	Sequence 3
Sequence 3	Sequence 4
Sequence 4	Sequence 5
Sequence 5	Sequence to be predicted

Figure 3.7: Forming feature and target sequences for $N = 10$ and $S = 5$

sequences for the case N=10 and S=5 in Figure 3.7. This connectionist technique learns effectively the patterns as the values change with frequency in the signal. Connection based methods generally exploit the interrelationships between variables. It can generate complex representations about its environment. In this context, the LSTM-RNN is a connectionist learning technique because it finds a a reproducible representation among frequency indices. This learned representation is then used to extrapolate the response. The LSTM-RNN network consists of a deeper hidden layer network that takes in a batch size of G frequency sequences in sequential order with total of B batches. For each batch

of the input, the LSTM layer produces the future values of size F samples. Hence, during the training phase, this action is iterated over all the batches and the mean squared error is computed between the real part and the predicted real part. Once the training is done, the inference phase follows. During the inference phase, the feature sequence has frequency samples from index $N - S$ to N and the target sequence has the $N - S + 1$ to $N + 1$ - to predict the future sample. We do this F number of times to infer the F future samples. Since there is no training data in this phase, we use our predictions to form the next set of sequences.

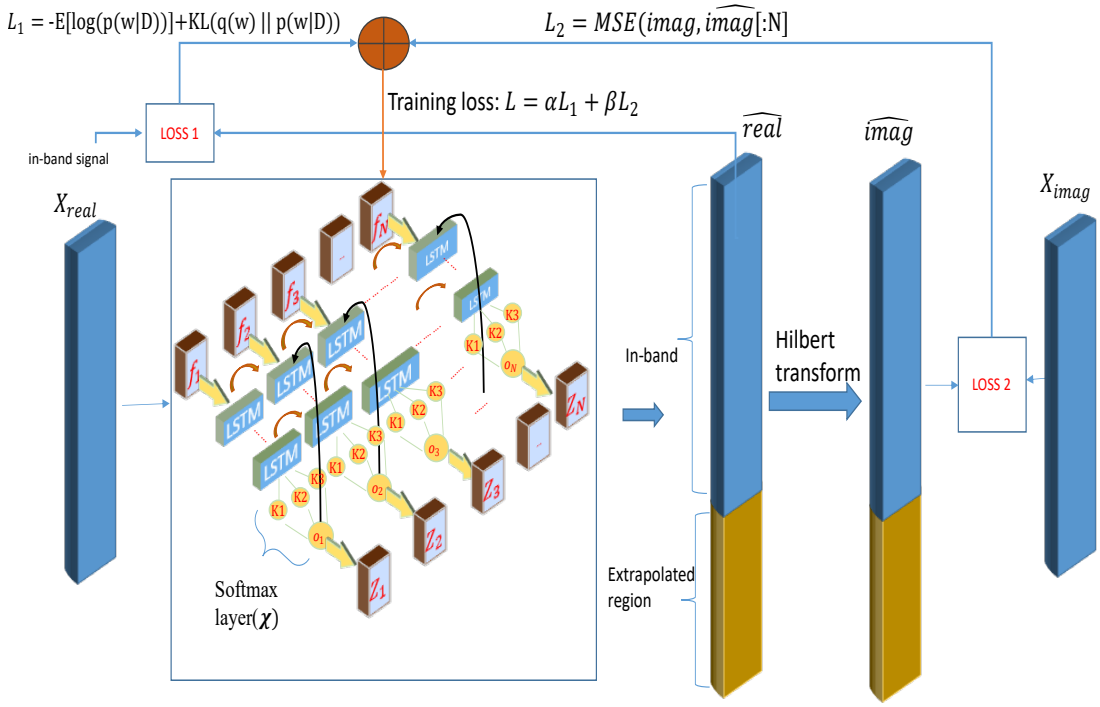


Figure 3.8: HilbertNet architecture:(i) in-band real part is fed to the LSTM_RNN network to get probabilistic output including in-band and out-of-band part, (ii) Hilbert transform of the real part is taken to obtain imaginary part, (iii) Losses are computed between the predicted real and imaginary real, and the actual real and imaginary parts respectively. (iv) the model parameters are updated through gradient descent after combining the losses

Traditionally, the outputs of a Neural Network (NN) are deterministic. The challenge with deterministic NNs is that the prediction is exact without any estimate of the probable error around predictions. This often gives an incomplete picture of the system response because in actual design, one may never have the extrapolated response to compare it to.

It is therefore important to quantify the uncertainty around prediction instead of the usual error backpropagation for training the NNs. To address the uncertainty involved in our predictions, we make use of the Bayes by Backprop (BBB) algorithm applied to RNNs [43]. For such a purpose, instead of assuming variable deterministic parameters of the **W**s and **b**s matrices as in equations (3.3)-(3.6), we assume they are samples from a prior distribution. The output Z is then a probability distribution that is computed using the given the independent frequency data input D. Hence, we marginalize the probability over all the various weights to get the distribution of outputs given the inputs as:

$$p(Z|D) = \int_{\theta} p(Z|D, \theta')p(\theta'|D)d\theta' \tag{3.13}$$

where θ represents the set of the parameters of the weights distribution of the model and θ' is an iterator over it. The second term in equation (Equation 3.13), $p(\theta|D)$, is intracbecause it marginalizes over all the model parameters i.e. every possibility of models which is impossible to calculate analytically given finite compute time and resources. Consider an arbitrary distribution $q_{\phi}(\theta)$, parameterized by ϕ which is chosen from the set of known distributions. The goal is to find the parameters of $q(.)$ such that the two probability distributions are similar. Therefore, we minimize the Kullback-Leibler (KL) divergence between the two distributions. The KL divergence between two density functions $f(.)$ and $g(.)$ is defined as:

$$KL[f(x)||g(x)] = \int f(x)log(\frac{f(x)}{g(x)})dx \tag{3.14}$$

Hence, the optimum set of parameters θ_{opt} is chosen such that KL divergence is minimized given by:

$$\theta_{opt} = \arg\min_{\theta} KL[q_{\phi}(\theta)||p(\theta|D)] \tag{3.15}$$

We can express the conditional distribution of the model parameters with respect to the

given data through the Bayes' rule:

$$p(\theta|D) = \frac{p(D|\theta) * p(\theta)}{p(D)} \tag{3.16}$$

The prior $p(\theta)$ is the set of parameters to which we believe could have generated our predictions in the extrapolated space. It represents the parameters set is supposed to look like before training. When the model discovers a new batch of training data, the network makes an informed judgement about the parameters by computing the posterior $p(\theta|D)$. To enable this process we initially assume standard gaussian priors. The model parameters converge to their true posterior distribution after training. The approximate variational distribution, $q_\theta(w)$ is chosen from a set of known distributions. Traditionally it is chosen as a mixture of gaussians whose means are Bernoulli distributed. The loss function using the parameters can be defined using:

$$L_1(\theta) = -E_{q(\theta)}\{logp(Z|\theta, D)\} \tag{3.17}$$

The loss function is the variational free energy which we have to minimize to achieve the "Evidence Lower Bound". To harness gradient descent for a probabilistic model, we use the local reparameterization trick [43], also shown in Algorithm algorithm 2. A standard gaussian distribution is sampled $\epsilon \sim \mathcal{N}(0, 1)$. A model parameter, the elements of the parameter space, w is then computed as $w = \mu + \epsilon * \sigma$. For training, the update rule now applies to the mean and standard deviation for each parameter for a given batch as indicated in steps 11 and 12 of Algorithm algorithm 2. Therefore, the output of such the LSTM-RNN is now a distribution computed using marginalization of the probability distributions of the weights and biases, given the input sequences. As the output is a distribution, its mean and variance are known. The mean is represented as the predicted output and the variance is used to calculate the upper and lower confidence bounds around the predicted mean thus, quantifying the uncertainty in the prediction or the error bar around the prediction.

The importance of uncertainty quantification was highlighted in the previous chap-

Algorithm 2: Training Hilbert-RNN architecture

Input: $Z_{in-band}(f)$, nepochs, dynamic learning rate
Output: $\hat{Z}_{out-of-band}(f)$ with 95% confidence

1: Split $Z_{in-band}(f)$ into $y_r = Re[Z_{in-band}(f)]$ and $y_{imag} = Imag[Z_{in-band}(f)]$
2: Adaptive Windowing operation on y_r and MinMax scale according to eqn. (Equation 3.12).
3: Form feature input and target output sequences top form a sequence set
4: Make B batches for G sequence set each
5: Sample $\epsilon \sim \mathcal{N}(0, 1)$. $\mu + \epsilon * \sigma$
6: Feed feature input sequences into LSTM-RNN with parameters θ to predict target sequences for all batches to give output y_r
7: Measure Loss between real in-band part and predicted real in-band part : $L_1 = loss(y_r, \hat{y_r}[: N])$
8: Form extended real-valued response $y_{real} = [y_r, y_r[N : \hat{N} + F]]$
9: Compute Hilbert transform: HT(y_{real}) = $\hat{y}_{imag}$
10: Compute Loss: $L_2 = MSE(\hat{y}_{imag}[: N], y_{imag})$
11: Combine losses: $L = \alpha * L_1 + \beta * L_2$
12: Compute gradient of loss $g = \partial L / \partial \theta$
13: $\mu \leftarrow \mu - \eta * \frac{g}{B}$
14: $\sigma \leftarrow \sigma - \eta * \frac{g}{B}$
15: Repeat steps 4 through 12, till nepochs reached
16: $Z_{out-of-band}(f) = \hat{y}_{real} + j * \hat{y}_{imag}$ =0

ter. Frequency spectrum extrapolation with uncertainty estimates can be performed using Bayesian RNN as described in [44]. By creating a unique wrapper around a Bayesian RNN, a NN called "Hilbertnet" is created. In the Hilbertnet, a Hilbert transform is used to generate a causal complex-valued extrapolation with 95% confidence region around predictions.

Input to the Hilbertnet, shown in Figure 3.8, is the in-band complex system response. This is then passed to the Bayesian LSTM RNN. The Bayesian LSTM RNN extrapolates the real-valued system response. Extrapolation of the imaginary part is obtained using the Hilbert transform of the real part. Since the real response is finite in frequency, the Hilbert transform becomes an approximation. Therefore the extrapolation introduces two types of errors here; (1) the error between the real response produced by the LSTM-RNN and (2) the error associated with the imaginary response produced by the Hilbert transform and the imaginary in-band response. These two responses are combined to train the network

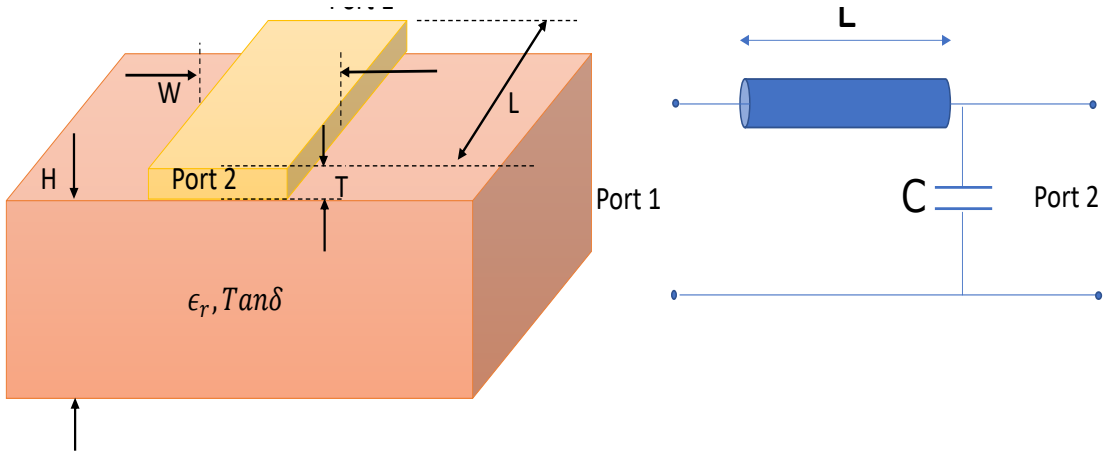


Figure 3.9: (a) Microstrip line (b) Transmission line circuit

through BBB, thus enabling complex-valued causal extrapolation in the system frequency response.

3.4 Numerical Example 1: Microstrip Circuit

As a first example, we apply our approach to a transmission line model as shown in Fig. Figure 3.9. A shunt capacitor is attached at the end of the microstrip. As expected, this 2-port system has a decaying insertion loss with increasing frequency with periodic ripples on it. The circuit parameters are shown in Table Table 3.1. We are interested in extrapolating the complex-valued return loss (S_{11}) and insertion loss(S_{21}). Frequency response data is collected by simulating the circuit in a typical full-wave simulator from DC to 50GHz - a total of 1500 points.

We use 750 of the data samples for the training phase from frequency 0GHz to 25GHz and the other 750 for evaluation phase from frequency 25GHz to 50GHz. In this example, we use 2 hidden layers for the LSTM network containing 10 and 15 unit cells respectively. The results for the model are shown in Fig. Figure 3.10 that depict the comparison of real and imaginary parts of the return and insertion loss.

We see that there is an oscillating response exhibited by the real and imaginary part

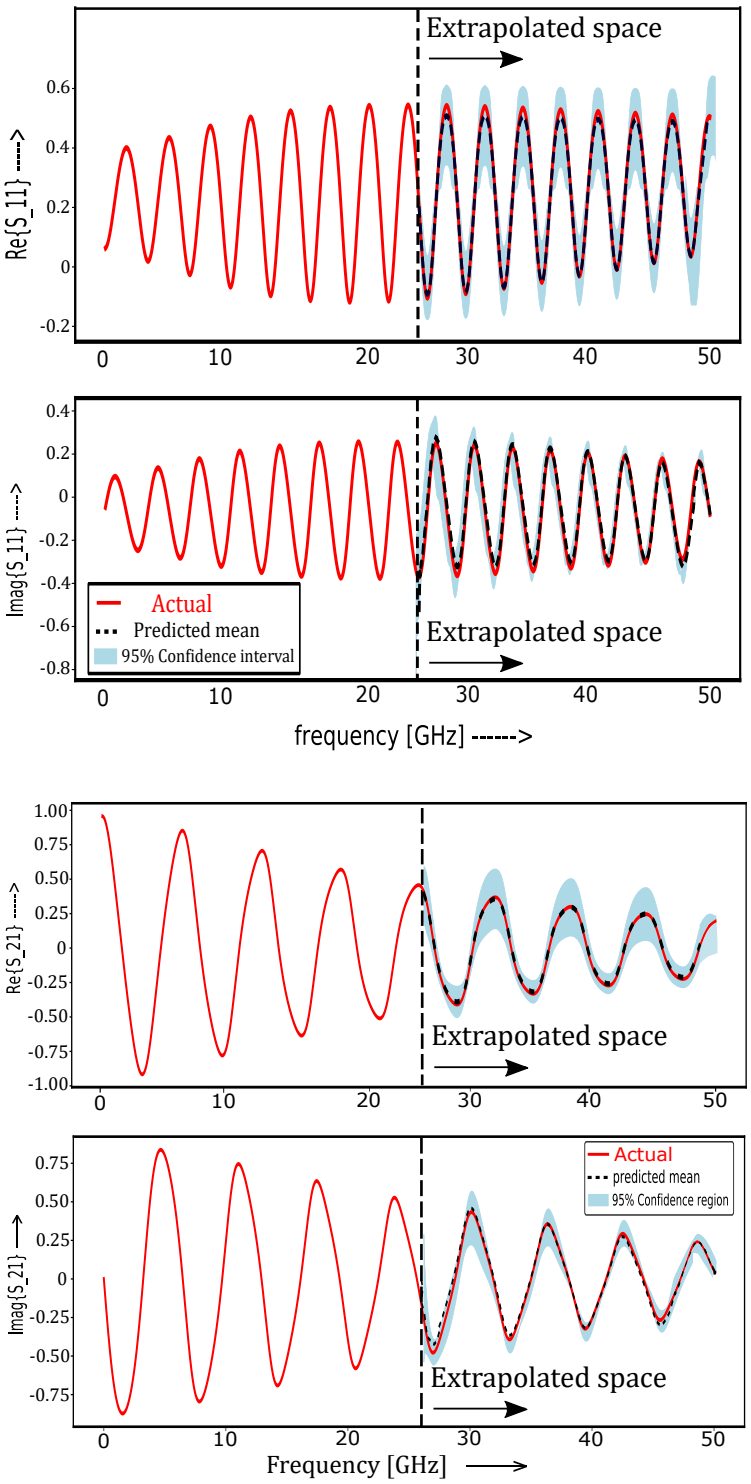


Figure 3.10: Insertion Loss and Return Loss of microstrip circuit

Table 3.1: Microstrip circuit parameters

Parameters	Symbol	Value
Conductivity	σ	5.6 x 10^7 S/m
Metal height	t_{metal}	25.4 μm
Width	W	406 μm
Length	L	254 mm
Loss Tangent	$Tan\delta$	0.02
Capacitor	C	0.2 pF
Design Impedance	Z_o	50 ohms
Dielectric Thickness	H	508 μm
Dielectric permittivity		4.5 ϵ_o

of insertion loss S_{21}. The cutoff frequency is 25GHz. The network is able to learn the sinosoidal trend. The patterns of change are remembered by the LSTM cells and hilbert transform is able to recover the imaginary part from the real part since the signal is analytical. We also show the confidence intervals for our predictions that cover 2 standard deviations from the mean prediction. At the amplitude peaks, the prediction interval increases depicting the realiability of the results at that frequency.

3.5 Numerical Example 2: Co-planar waveguide

We show our extrapolation results with co-planar waveguides using measured data as reference. A CPW, 4mm long, is fabricated on a glass substrate bounded with polymer making a 3-layer stack up as shown in Fig. Figure 3.11. The polymer material used here is the ABFGL-102 which is used on both sides of the glass substrate. The parameters of the CPW are shown in Table Table 3.2. Insertion and return loss (in dB) extrapolation is performed and results are compared with the measured results in D-band i.e. 110GHz to 170GHz. The training frequency range includes 110-140 GHz band whereas uncertainty and prediction evaluation is done from 140-170 GHz. For the CPW, we use two hidden layers with

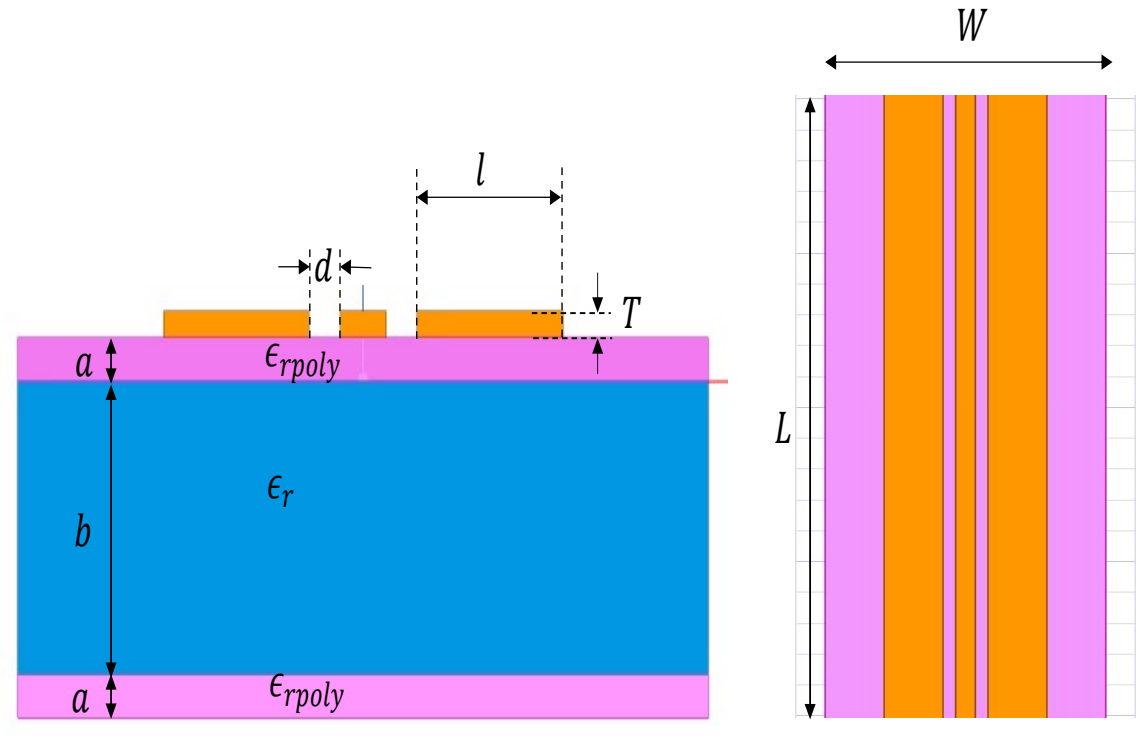


Figure 3.11: (a) Stack up (b) Top view of coplanar waveguide

10 and 15 LSTM unit cells respectively. The root mean squared loss between the mean of the predicted frequency sample and the actual value is 10^{-2} dB. For this example, the given results are in magnitude so, it is 1D extrapolation and does not involve Hilbert transform. This algorithm relies on the interdependencies and trends among the waveguide since it is a periodic structure relative to the length of the waveguide. The results for the model are shown in Fig. Figure 3.12. Since, the model is trained on measured data, the confidence bounds and the predictions are not as smooth because it also captures the measurement noise effects.

3.6 Numerical Example 3: RF Filter

Consider an interdigital band pass filter from [**Ali˙RF˙filter**], shown in Fig. Figure 3.13. The filter response is centered around 28 GHz. The frequency points simulated are 450 total. In this example, we show how the confidence bounds change as we get more sam-

Table 3.2: Coplanar Waveguide parameters

Parameters	**Symbol**	**Value**
Conductivity	σ	5.6 x 10^7 S/m
Metal height	T	9 μm
Width	W	332 μm
Separation	d	15μm
Length	L	4 mm
Polymer height	a	15 μm
Glass thickness	b	100 μm
Metal width	l	70 μm
Design Impedance	Z_o	70 ohms
Poly Loss Tangent	$Tan\delta_{poly}$	0.044
Glass Loss Tangent	$Tan\delta_{glass}$	0.056
Poly-Dielectric permittivity	3.2 ϵ_o	
Glass-Dielectric permittivity	4.9 ϵ_o	

ples in the training range. The bandwidth and the peak insertion loss is controlled by the length and width of the microstrip stubs. The spacing between the stubs controls the coupling factor. We illustrate the effect of increasing N over the confidence bounds in Fig. Figure 3.14. Before training, that is $N = 0$, we have a uniform prior over the network parameters. Given new data to the model, the posterior is updated and provides new uncertainty estimates to adapt to the newly simulated points. The training points are uniformly distributed. At $N = 100$, the model is still unsure about its predictions since there is no information about bandpass response in the training data yet. At $N = 200$ and eventually $N = 300$, we see that the model, and the posterior is updated such as to minimize the KL divergence. Hence, the 95% confidence region shrinks from a wide uniform region to one that is centered around the mean. When the machine learning algorithm starts to train, the uncertainty is maximum because the model cannot be certain about the prediction since there is no training data. Thus, the weights of the model have some probability to go at any

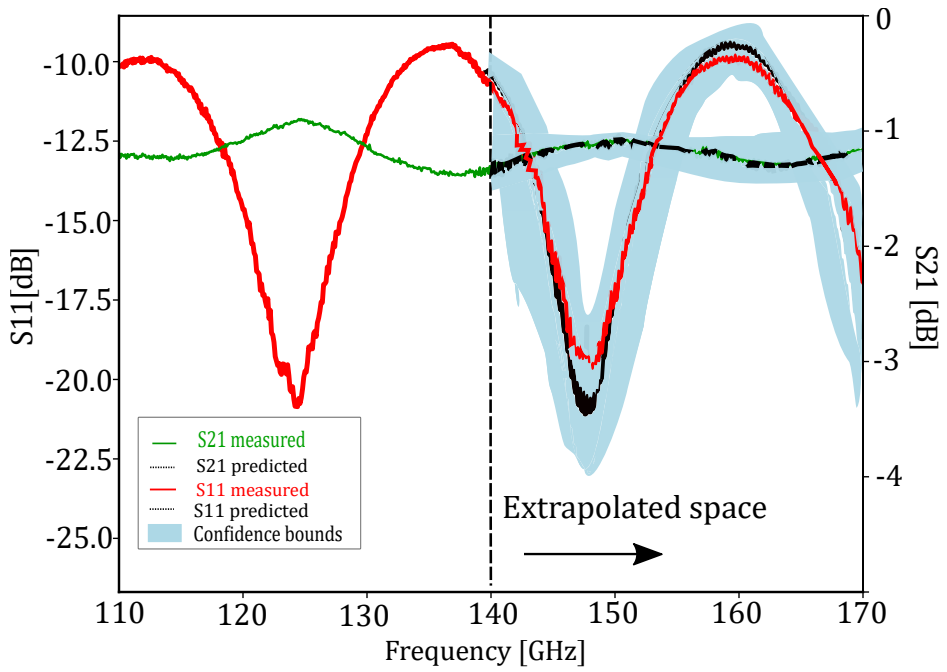


Figure 3.12: Insertion Loss and Return Loss of Coupled Waveguide in D-band

value. Thus $p(\theta)$ is standard gaussian. Once, some training data is provided to the model, the prior belief is being updated through the Bayes' by Backprop algorithm and the posterior function about the weights is shown as $p(\theta|D) \propto p(D|\theta)p(\theta)$. The variance of the posterior distribution is always lower than the prior because the model is updated. Thus, the model is more informed. The 95% confidence region which is about ± 2σ, shrinks from a larger value to a smaller one. The model parameters are now concentrated around their means closer to the true value. In this example, we use an LSTM network with 4 hidden layers with 15, 25, 32, 5 units respectively.

3.7 Numerical Example 4: Power Delivery Network

In power integrity applications, it is imperative to design the power delivery network (PDN) that exhibits minimal power supply noise which is dictated by the value of target impedance (Z_{target}). A typical PDN is illustrated in Fig. Figure 3.15. To design such systems, CAD

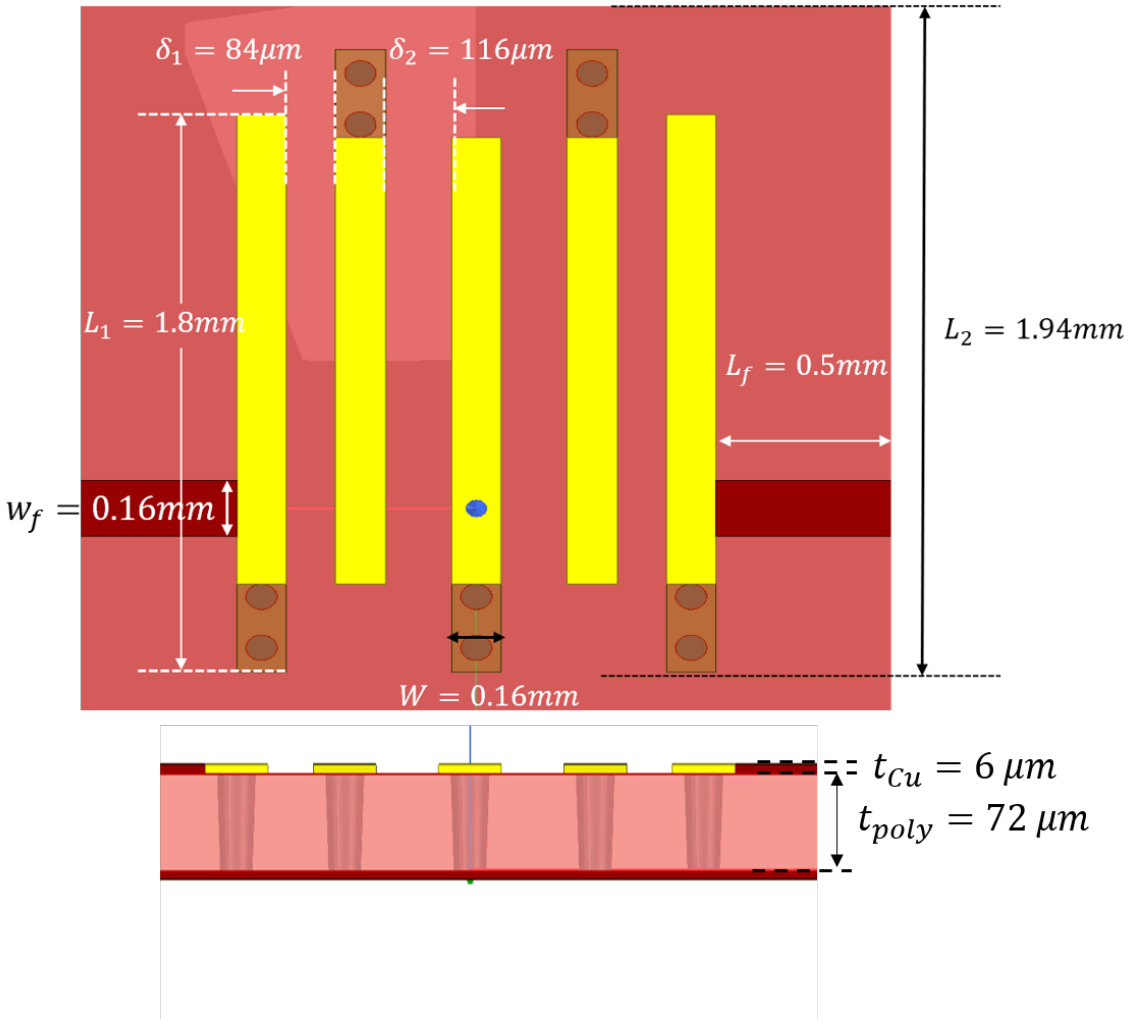


Figure 3.13: Fabricated 5th Order Interdigital Filter for 28GHz band

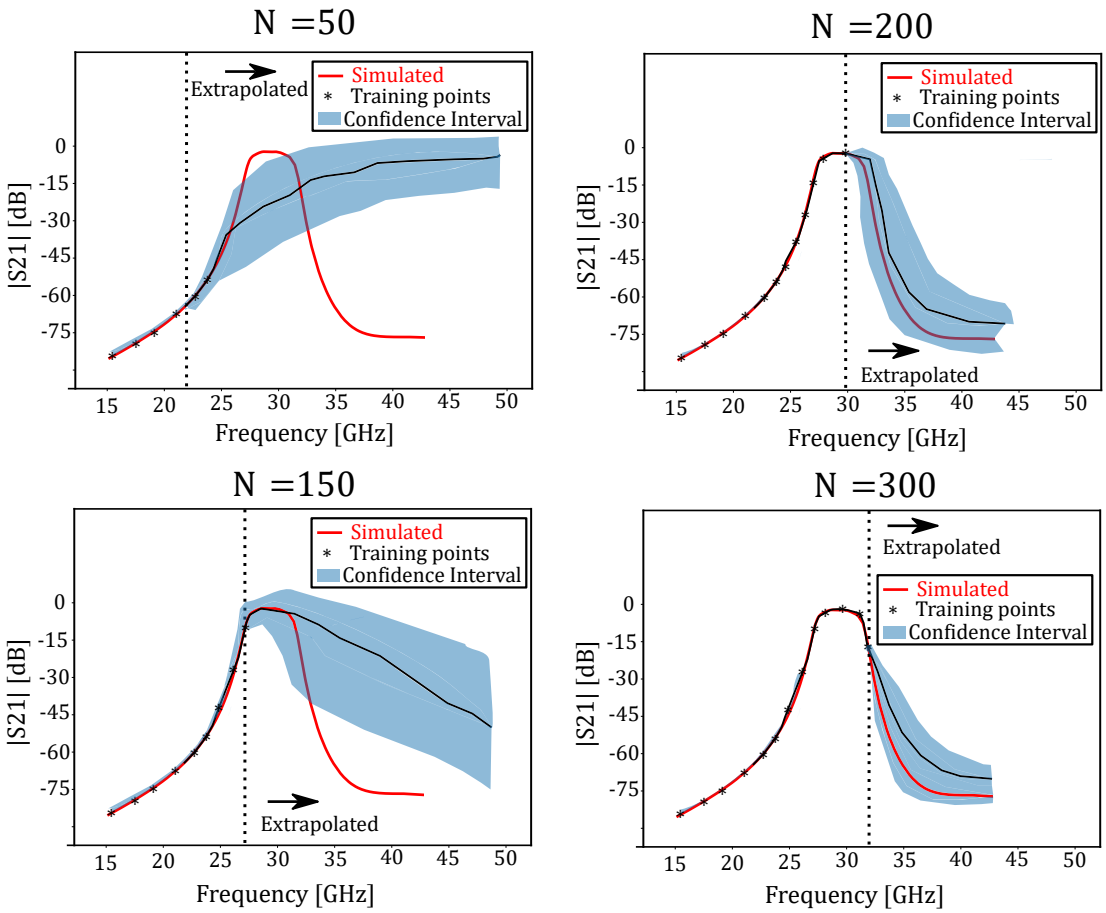


Figure 3.14: Uncertainty analysis for Insertion Loss for Interdigital filter

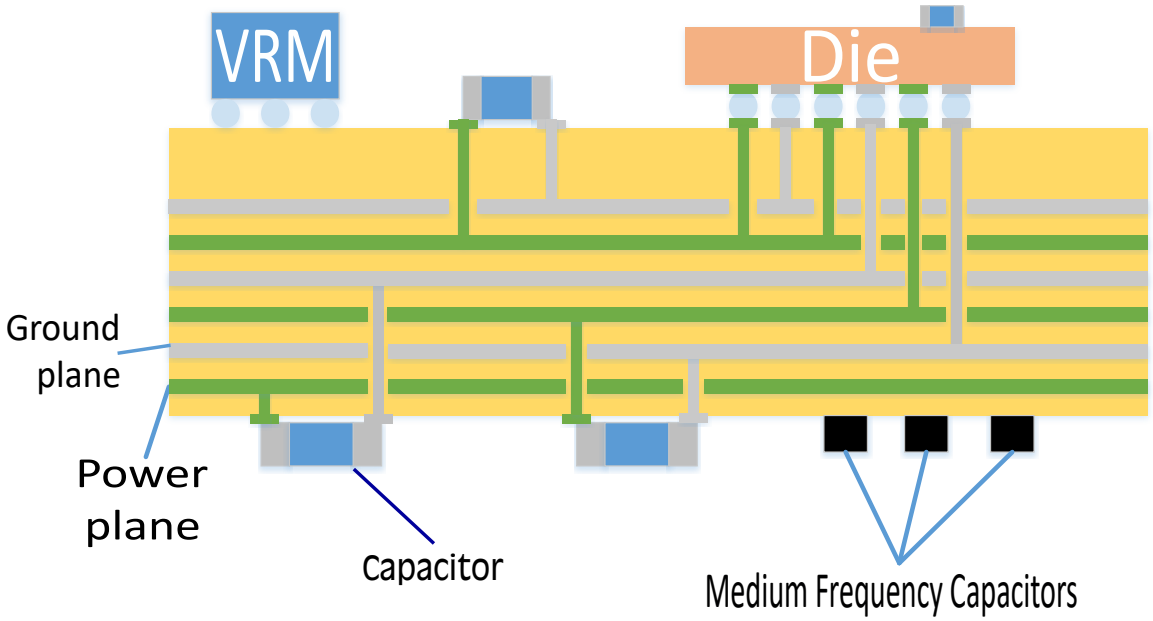


Figure 3.15: A typical power distribution network containing VRM, P/G planes, capacitor, C4 bumps, TSVs

tools and analytical techniques are harnessed to inspect and improve the design of PDN. The impedance response for power ground planes is derived in [45]. A segmentation method is used to estimate the impedance properties of the PDN in [46]. Extrapolating the frequency response is important since designers need to determine if there are any resonances in proximity to the modeled (or measured) data which is bandlimited.

We simulate the PDN using analytically derived formulae to compute the impedance response using analysis tools available online [**PDN3**] for training our model and comparing to it. Since the PDN consists of (i) interposer P/G grid (ii) PCB P/G plane (iii) C4/μ-bump array (iv) TSV array and (v) via array, the impedance from each component is calculated and then put together to give the PDN impedance (Z_{PDN}). The impedance response depends on the number, density and the placement location within the package containing the PDN. Since the PDN typically contains various different inductive and capacitive elements, one would expect the impedance response to have multiple resonant peaks and dips with sharp transitions because of inductive and capacitive effects. We divide the complex-valued

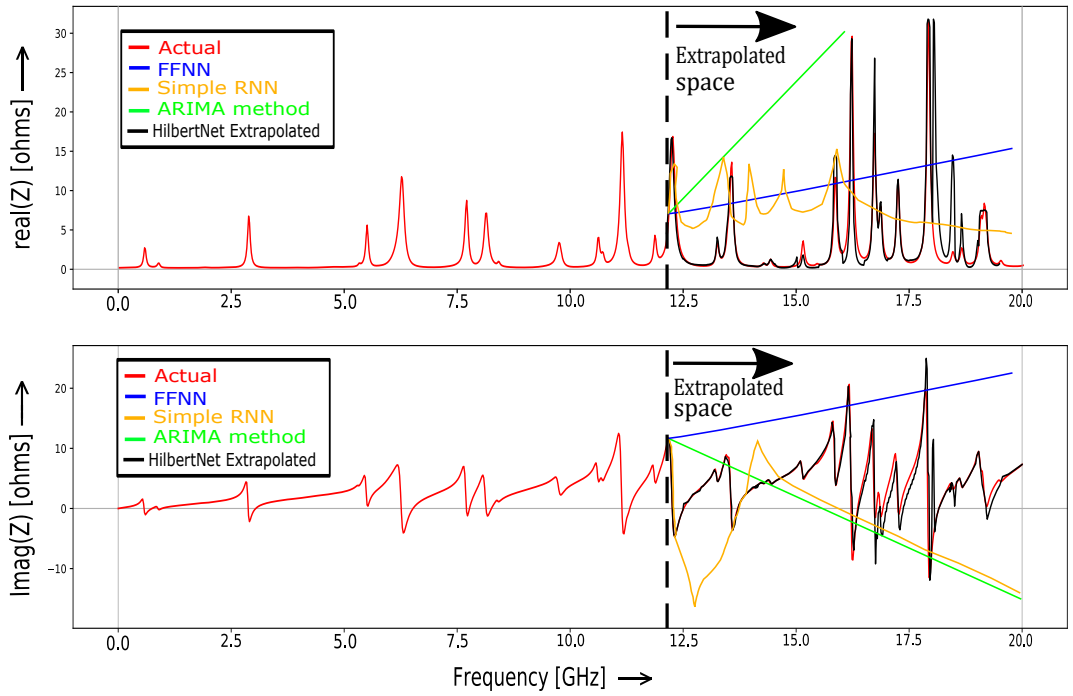


Figure 3.16: Impedance Response extrapolated

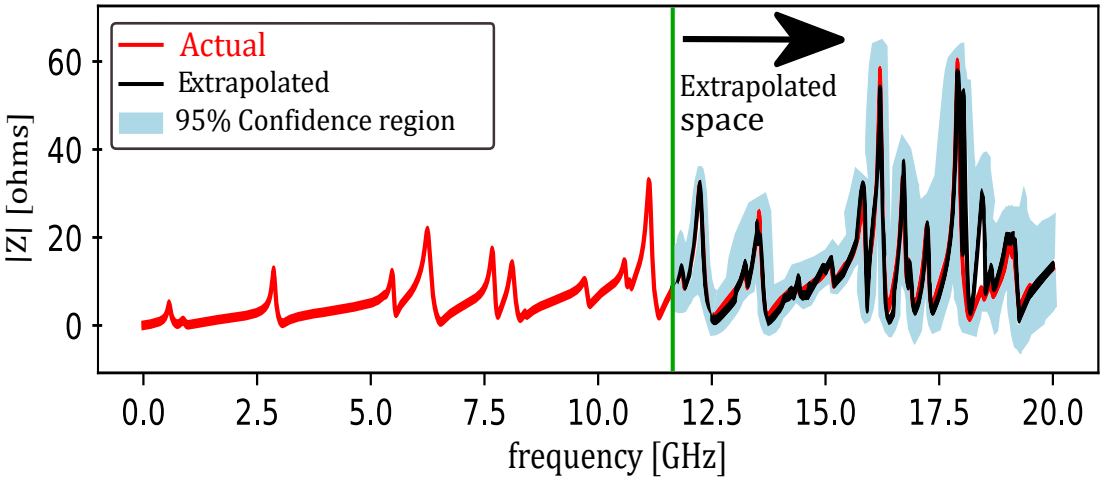


Figure 3.17: Absolute impedance response extrapolation with 95% confidence bounds

impedance into real and imaginary to feed into algorithm algorithm 2. The parameters characterizing the PDN are given in table Table 4.1. These values are chosen arbitrarily to provide the shown impedance response. Our approach should be generalizable to other

values as well.

Table 3.3: PDN parameters

Parameters	**Symbol**	***Value***
Conductivity	σ	9.27 x 10^7 S/m
Metal height	t_{metal}	0.757 μm
Grid width	W_{grid}	27 μm
Grid spacing	w_{grid}	168 μm
TSV radius	r_{TSV}	7.93 μm
TSV pitch	p_{TSV}	23.9 μm
C4 radius	r_{C4}	240 μm
C4 pitch	p_{C4}	0.72 mm
μ-bump radius	r_{μ}	12.125 μm
μ-bump pitch	p_U	45 μm
Substrate thickness	h_{imd}	0.81 μ m

For this application, we use 1000 frequency points divided into a 70/30 train/test split i.e. 700 linearly spaced frequency points ranging from 1 MHz to 12 GHz. For the LSTM-RNN, we employ 3 hidden layers consisting of 10, 20, 10 unit cells, respectively. The model uses dynamic adaptive learning rate trained by Adam optimizer through Pytorch package. We place a gaussian prior on parameters of the model to get confidence bounds on our predictions.

The quality of our prediction is measured by how close it predicts to the actual value of impedance. Fig. 17 shows the extrapolated real and imaginary response in comparison to the actual response. Here, we set the cuttoff frequency as 12 GHz beyond which is the extrapolated space. The predicted values follow the correct trend. Thus, we can predict a future pole given the information about the location and intensity of the past pole values without limiting the designer to the topology of the PDN explicitly. The predicted values are compared to the values obtained by other state-of-the-art techniques. We compare our methods to ARIMA models [23], a simple feed-forward neural network (FFNN) and a

simple RNN method.

When producing results with ARIMA, we use the optimal (p, q, d) design tuple where p is the order of the 'auto-regressive' term, q is the order of moving average. It is the number of lagged predictors the model takes into account while predicting the extrapolated value. An ARIMA model is given by:

$$y_t = \alpha + \beta_1 y_{t-1} + ... + \beta_p y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + ... + \phi_q \epsilon_{t-q} \tag{3.18}$$

where y_t is the current predicted value and the $\beta_i y_{t-i}$ terms denote the linear combination of lags upto p lags and the $\phi_i y_{t-q}$ terms denote the linear combination of lagged forecast errors upto q lags where $i = 1, ..., p$ or q appropriately. We optimize p, q for the given problem and obtained the results shown. The reason the ARIMA model is unable to even slightly follow the pattern is because the parameters extracted are not enough to determine the pattern. Such parameters are extracted once from the whole training set and there is no learning involved.

The results are compared next with a simple feed-forward neural network. The FFNN model learns the noise in the data but is unable to exhibit the pattern because it does not have any memory. The model used for FFNN is a 4-layer model containing [15,40,100,35] neurons each. Next, the proposed architecture is compared to a simple RNN. We do this to illustrate the need for specialized LSTM units inside a regular RNN to accurately extrapolate the response. We use a RNN with 3 layers having 10 recurrent cells each. We can see that the output of the RNN is able to vary itself given the training range. However, the range of the output is incorrect. This is because the RNN does not understand which patterns are important and which patterns need to be forgotten by the network. Hence it is unable to retain information for longer duration far off from the training sequence.

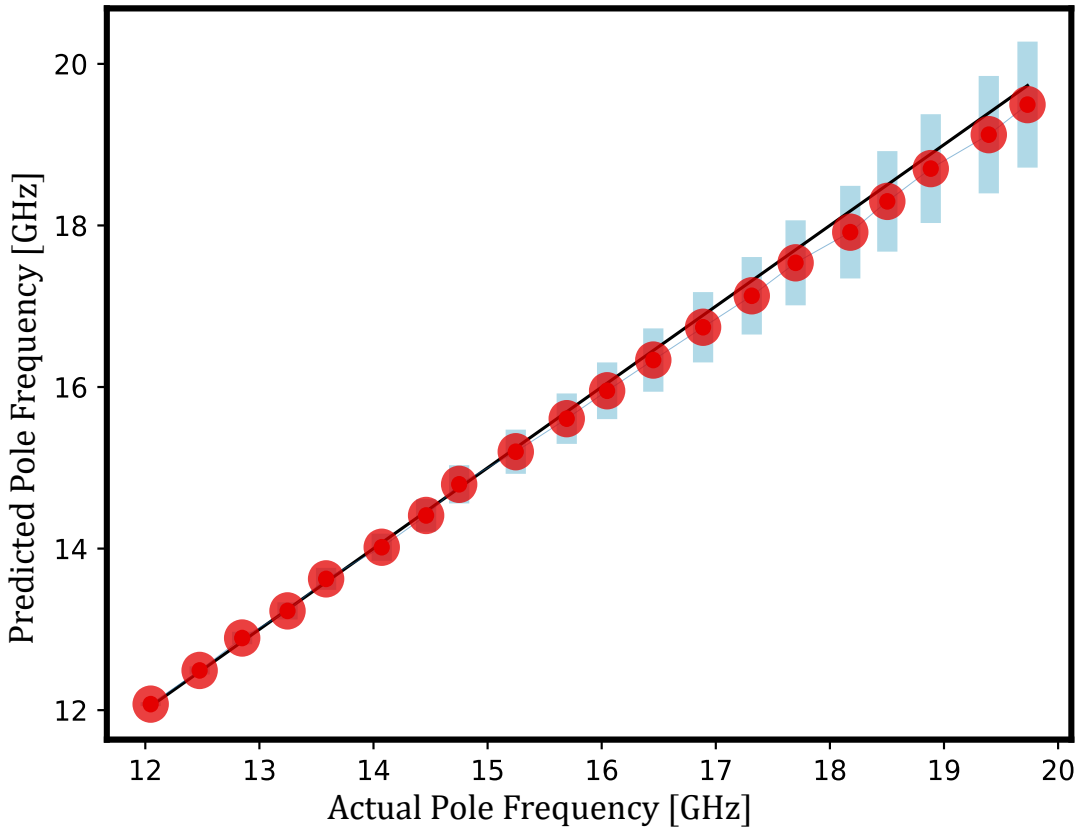


Figure 3.18: 95% confidence intervals versus cutoff frequency (f_c)

3.7.1 Discussion on Extrapolation Range

Using the BBB for RNNs, we estimate model uncertainty in Fig. Figure 3.17. We plot the absolute impedance response against frequency. The designer is provided with the upper and lower bounds to assess the reliability of the predictions. This enables the designer to simulate in the vicinity of the pole where the prediction is less confident as opposed to the tighter bound. We also observe from the Fig. Figure 3.17 that as the predictions go far off from the cutt-off frequency point, the model naturally becomes less confident. This is expected since the model has longer path to backpropagate and KL divergence loss increases. We investigate this effect in Fig. Figure 3.18 where uncertainty estimates are compared with actual versus predicted value. The closer the measurement is to the identity line the more accurate the prediction is. Fig. Figure 3.18 shows the errorplot for the case where the

cutoff $f_c = 12GHz$. It can be observed that till 16GHz the predictions are almost exact. However, when we move on to higher frequencies, the variance of the prediction becomes higher indicating that the model is less sure about those predictions. This is expected since the training range of frequencies was way past that point. At 20GHz, the confidence bounds grows loose. To make tighter bounds at higher frequencies, one way is to increase the cut-toff frequency. This will not only add more training data and points but will make the model comprehend more intricate seasonal and local periodicities. The composite loss considers the two component losses, (1) coming from the LSTM-RNN marginal log likelihood (2) coming from the error in hilbert transform as indicated in Section IV. Here, $\alpha = 0.5$ is used as an unbiased weight distribution from both sections of the network.

3.8 Computation time and cost

For different examples, different amounts of computational resources were harnessed. All the examples were run on a windows machine with 16GB RAM and processor Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz. We are working with Python3.7.6 and Cuda toolkit v11.3.1. The GPU used was NVIDIA GeForce GTX 1050 with Max-Q Design. We list the computational cost for each example in table Table 3.4. We compare the computational time required for training and evaluation of HilbertNet with a simple RNN mentioned above. It can be observed that the time difference is not stark but accuracy for the HilbertNet is higher.

For more details, the code is available here:
https://github.com/owbhatti/hilbertnet_frequency_extrapolation.

3.9 Conclusion

In summary, a method for causal extrapolation of complex-valued frequency response for distributed structures is presented. The technique uses hidden dependencies in the response to extrapolate it beyond the in-band range. We use recurrent neural networks to learn

Table 3.4: Time resources

Example	**HilbertNet**		**Simple RNN**	
	Training	Evaluation	Training	Evaluation
Microstrip Structure	20 min	10 ms	15min	10ms
Co-planar waveguide	23 min	8 ms	18min	7ms
RF filter	18 min	7 ms	15min	6ms
Power Delivery Network	34 min	17 ms	26min	16ms

these dependencies. The hilbert transform layer creates a relation from the real part to the imaginary part, ensuring that the extrapolation is causal. The technique is applied to four scenarios (1) microstrip line circuit (2) Co-planar waveguide measured results (3) Interdigital bandpass filter and (4) Power delivery network, with the last being the most challenging to extrapolate. The PDN provides an accuracy of 0.008 ohms. To address model uncertainty, we employ Bayesian RNN that provides confidence. A quantitative study of cutoff versus confidence bound is performed The model becomes less confident with less training data and in predictions where the frequency points is far off from the cutoff frequency. Furthermore, for a given error, it is possible to extrapolate upto a certain range given by the variance of the predictions.

CHAPTER 4

DESIGN SPACE EXTRAPOLATION NEURAL NETWORK

4.1 Problem Statement

Traditional approaches include deriving analytical relations to output the frequency response by lumped modeling of circuits which can then be used to perform extrapolation in design space. However, such approaches are not accurate since they involve lumped approximations that do not include the effect of parasitics at higher frequencies. To model the non-linear mapping, how complex they may be, from the input design space to learn the frequency response, it is essential to start from the distributed structure. Mathematically, it is desired to find a mapping F such that

$$Z(f) = F(\boldsymbol{X}, f) \tag{4.1}$$

where Z is the complex-valued frequency response as a function of operating frequency f and multimdimensional input design space parameter $X \in \mathcal{R}^D$ where D is the dimensionality of the input. Recently machine learning methods have shown to model non-linear mappings between inputs and outputs. Vanilla fully-connected neural networks provide a flatter response but lack spatial correlation between samples [47]. Since the input space is continuous, we make use of convolutional neural layers to find patterns between similar samples of data. However, this work is distinct in the sense that our aim is to predict the in-band response for the out-of-bound geometric and material design parameters. Following from equation (Equation 4.1), consider an out-of-bounds design tuple $X_{extrapolated}$. The goal is now to find $Z'(f)$ as a function of the frequency response of in-bound tuples, the

previous training tuples and the frequency:

$$Z'(f) = G(Z(f), \boldsymbol{X}_{\boldsymbol{extrapolated}}, f) \tag{4.2}$$

Our approach combines both the neural networks to form an assembly network for modelling the relationship between the frequency response and design parameters that not only functions on the distributed structure but also saves computational time and extensive simulations to be performed repeatedly while doing so. Our framework uses complex-valued frequency response as a correlated output having inter-channel correlations given the design tuple.

4.1.1 Transposed Convolutional Net Architecture

We propose an ensemble of two neural networks connected in series (1) Fully Connected Upsampler which increases the multidimensional design space into a latent space with a unique code for each sample and (2) Convolutional decoder consisting of transposed convolutional layers to compute the complex-valued frequency response for the power delivery network. The full network architecture is shown in Figure 4.1. This work is presented in [48].

The upsampler features a fully connected neural network containing linear layers with each layer having more number of outputs as compared to that of inputs. The design space having dimension D acts an an input to the upsampler and a latent code is in L-dimensional space. The code produced as an output of the network contains the unique feature vector to describe the response of the network as a function of the geometrical and material parameter values. Such an upsampler is depicted in Figure 4.1. For such an architecture, the output code is given

$$S = g_1(g_2(...g_h(x))....) \tag{4.3}$$

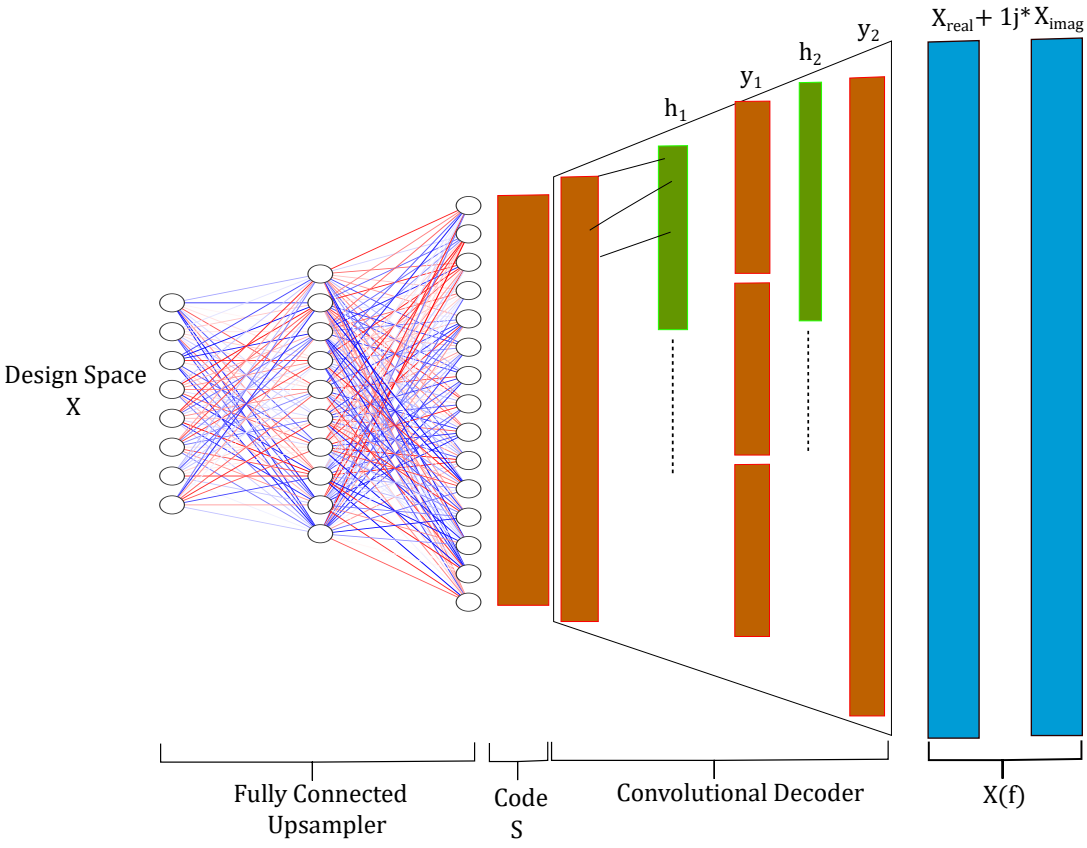


Figure 4.1: Proposed Network Architecture

where g_i is the transformation with activation function included from the ith to $i+1$th hidden layer with a total of h layers.

The convolutional Decoder consists of hidden transposed convolutional layers. A convolutional layer exploits the spatial correlations among the input vector for accurate feature representation. A trained transposed convolutional network captures the hidden dependencies in the data. In a vanilla convolutional neural network, output H of each layer l is a cross-correlation product of the input I with a sliding kernel K given in the following equation:

$$H_{i,j} = r((I * K)_{i,j}) = r(\sum_{m} \sum_{n} K_{m,n} * I_{i-m,j-n}) \tag{4.4}$$

where i, j, m, n are non-negative integers spanning the input space and r(.) is the non-linear activation function. The kernel K here plays a crucial role since, the output depends on how the kernel is slid on top of the input plane. The specificity, stride and padding decide the output dimensionality. Usually the kernel is designed as to decrease the dimensionality of the output. To upsample the latent design frequency space to produce the actual whole frequency response in the complex-domain, we make use of transposed convolutional blocks [49]. Such blocks enable higher output dimensionality by using a transposed kernel G. Again, we can write the convolution operation as

$$\begin{pmatrix} \\ H \\ \\ \end{pmatrix} = \begin{pmatrix} \\ K \\ \\ \end{pmatrix} * \begin{pmatrix} \\ I \\ \\ \end{pmatrix} \tag{4.5}$$

but transposed convolutional layer has the kernel matrix transposed

$$\begin{pmatrix} \\ H \\ \\ \end{pmatrix} = \begin{pmatrix} \\ G \\ \\ \end{pmatrix} * \begin{pmatrix} \\ I \\ \\ \end{pmatrix} \tag{4.6}$$

The output size of the transposed layer is given as

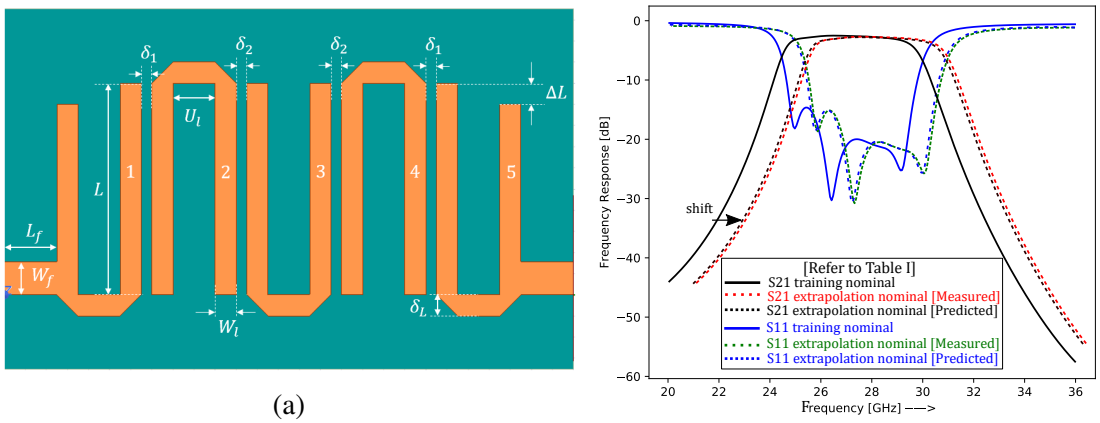
$$D_{output} = \frac{D_{input} + 2 * padding - (D_{kernel} - 1) - 1}{stride} + 1 \tag{4.7}$$

where D_x is the dimensionality of any vector **x**. Such an architecture is shown in Fig.**??**

4.2 Numerical Example 1: RF filter

A fifth order hairpin band-pass filter is shown in Figure 4.2a, from [50]. The filter response depends on the length and width of the stub. The various design parameters of this hairpin filter are listed in Table 4.1 along with their nominal design values. A DSENN is trained using a range of design parameters and material properties around the nominal value as shown in Table 4.1. The model is then tested using the values of design parameter and material properties listed in the extrapolation space shown in Table 4.1. The predicted frequency response of the nominal design in the extrapolation range is compared to that of a full-wave 3D EM solver in Figure 4.2b. As expected, the nominal design in the extrapolated space shows a distinct right shift in the cut-off frequency. The plot in Figure 4.2b indicates very good correlation between the measured and the predicted data. For this example, we use 140 design tuples for training and 100 designs for testing.

The next example is a second order Substrate Integrated Waveguide (SIW) filter presented in [51]. The filter structure and dimensions are shown in Figure 4.3a. The design



(a) (b)

Figure 4.2: Fifth order Hairpin BPF Ring Resonator (a) structure and (b) design space extrapolation results [refer to Table 4.1]

Table 4.1: Fifth order Hairpin filter parameters [Dimensionality=8]

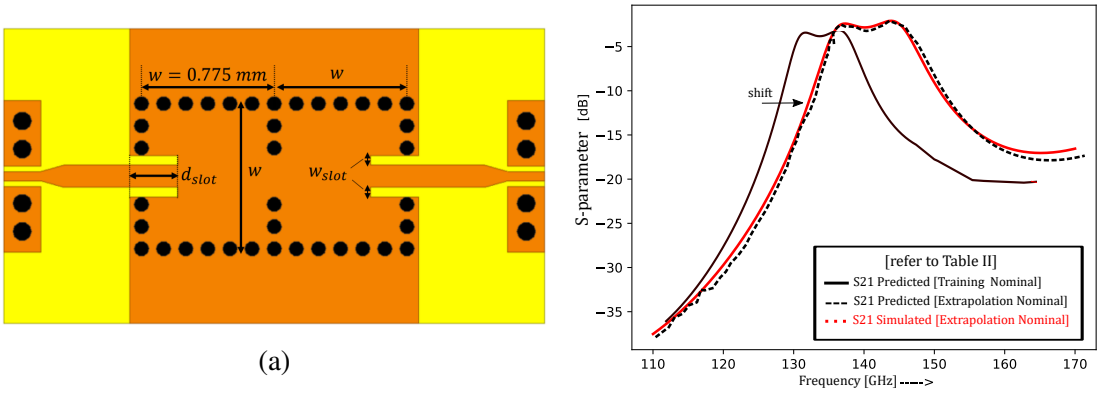
Parameters	**Symbol**	**Training Range**			**Extrapolation Space**		
.	.	***Min value***	***Nominal value***	***Max value***	***Min value***	***Nominal value***	***Max value***
Feed Length	L_f	150 μm	250 μm	250 μm	250 μm	275 μm	280 μm
Feed width	W_f	80 μm	100 μm	110 μm	110μm	157 μm	180 μm
Stub length	L	750 μm	770 μm	850 μm	850 μm	1000 μm	1100 μm
Stub width	W	50 μm	65 μm	75 μm	75 μm	100 μm	100 μm
Delta	ΔL	100 μm	100 μm	100 μm	100 μm	100 μm	100 μm
Stub gap	U_l	50 μm	50 μm	55 μm	55 μm	57 μm	60 μm
Gap 1	δ_1	40 μm	45 μm	50 μm	50 μm	55 μm	60 μm
Gap 2	δ_2	40 μm	45 μm	50 μm	50 μm	55 μm	60 μm

parameters are listed in Table 4.2. The filter response is a complex function of the design space shown in Table 4.2. In Table 4.2, the nominal design values as well as the training and extrapolation range are listed. For this example, we use 140 design tuples for training and 100 designs for testing. The filter response for the nominal design in the extrapolation range using the ML model is shown in Figure 4.3b. The training of the CNN based model is done using latin-hypercube sampling (LHS) of the design parameters in the training range [52]. LHS sampling is a way for generating almost random samples from the multidimensional distribution. It can be seen from Figure 4.3b that the CNN method has a

Table 4.2: Second order SIW filter parameters [Dimensionality = 11]

Parameters .	**Symbol** .	**Training Range**			**Extrapolation Space**		
		Min value	***Nominal value***	***Max value***	***Min value***	***Nominal value***	***Max value***
Transition width	w_t	0.1 mm	0.12mm	0.3mm	0.3mm	0.45mm	0.5mm
CPW width	w_{CPW}	20 μm	50 μm	70 μm	70 μm	75 μm	100 μm
CPW g-plane	g_{CPW}	10 μm	30 μm	45 μm	45 μm	50 μm	50 μm
Resonant width	w_{res}	600 μm	775 μm	800 μm	800 μm	810 μm	900 μm
Via edge-to-edge distance	s	50 μm	50 μm	55 μm	55 μm	57 μm	60 μm
Metal height	t_{Cu}	5 μm	9 μm	10 μm	10 μm	13 μm	20 μm
poly height	t_{poly}	50 μm	70 μm	150 μm	150 μm	180 μm	250 μm
Via diameter	d_{via}	100 μm	100 μm	105 μm	105 μm	108 μm	110 μm
Microstrip width	w_{MS}	50 μm	120 μm	120 μm	120 μm	300 μm	500 μm
Slot Depth	d_{slot}	40 μm	50 μm	60 μm	60 μm	85 μm	120 μm
Slot Width	w_{slot}	150 μm	175 μm	200 μm	200 μm	210 μm	300 μm

good prediction accuracy.



(a)

(b)

Figure 4.3: Second-order SIW filter (a) structure and (b) design space extrapolation results [refer to Table 4.2]

4.3 Numerical Example 2: Power Delivery

The PDN characterization geometrical and material parameters are stated in Table Table 4.1. We divide our design space into training and extrapolation space. To gather train and test data, we make use of online available tool for PDN impedance analysis [11]. We

Table 4.3: PDN characterization parameters

Parameters .	**Symbol** .	**Training Range**			**Extrapolation Space**		
		Min value	***Nominal value***	***Max value***	***Min value***	***Nominal value***	***Max value***
Conductivity	σ	1 x 10^7 S/m	5 x $10^7 S/m$	8 x 10^7 S/m	8 x 10^7 S/m	9 x 10^7 S/m	10 x 10^7 S/m
Metal height	t_{metal}	0.5 μm	0.8 μm	0.8 μm	0.8 μm	0.95 μm	1 μm
Grid width	W_{grid}	10 μm	15 μm	20 μm	20 μm	27 μm	30 μm
Grid spacing	w_{grid}	100 μm	125 μm	200 μm	200 μm	250 μm	300 μm
TSV radius	r_{TSV}	5 μm	10 μm	15 μm	15 μm	20 μm	25 μm
TSV pitch	p_{TSV}	15 μm	35 μm	60 μm	60 μm	70 μm	75 μm
C4 radius	r_{C4}	50 μm	100 μm	175 μm	175 μm	200 μm	275 μm
C4 pitch	p_{C4}	0.15 mm	0.5 mm	0.5 mm	0.5 mm	0.6mm	0.75 mm
μ-bump radius	r_μ	10 μm	10 μm	12 μm	12 μm	15 μm	15 μm
μ-bump pitch	p_U	40 μm	42 μm	45 μm	45 μm	47 μm	50 μm
Substrate thickness	h_{imd}	0.7 μm	0.7 μm	0.85 μm	0.85 μm	1 μm	1 μm
Si-dielectric	ϵ_{Si}	11.9 ϵ_o					
Poly-dielectric	ϵ_{poly}	3.9 ϵ_o					

perform a sensitivity analysis for the PDN to obtain insight into how frequency poles move with changing parameters. For example, we conclude that small changes in parameter W_{grid} i.e. grid width are responsible for larger changes in the impedance response as compared to other parameters. This effect is depicted in Figure 4.4. Furthermore, the change in response is greater in high frequency as compared to low frequency. This is because of the various parasitics that take effect at higher frequency as the inductances have a more pronounced effect ($|Z_L| = wL$ where w is the angular frequency and L is inductance). As grid width increases, the path from input to output on the plane increases which , in turn, affects the plane impedance. It introduces multiple current return paths resulting in multiple poles moving to higher frequencies.

In Table 4.3, the nominal value is shown which is arbitrarily picked to show that the network able to interpolate. We train the network on 1000 design tuples with each design tuple having its respective complex-valued frequency response whereas the extrapolation space consists of 200 design points. An adaptive learning rate scheduler is employed to minimize the training time while achieving the minimum loss on the test set. We use

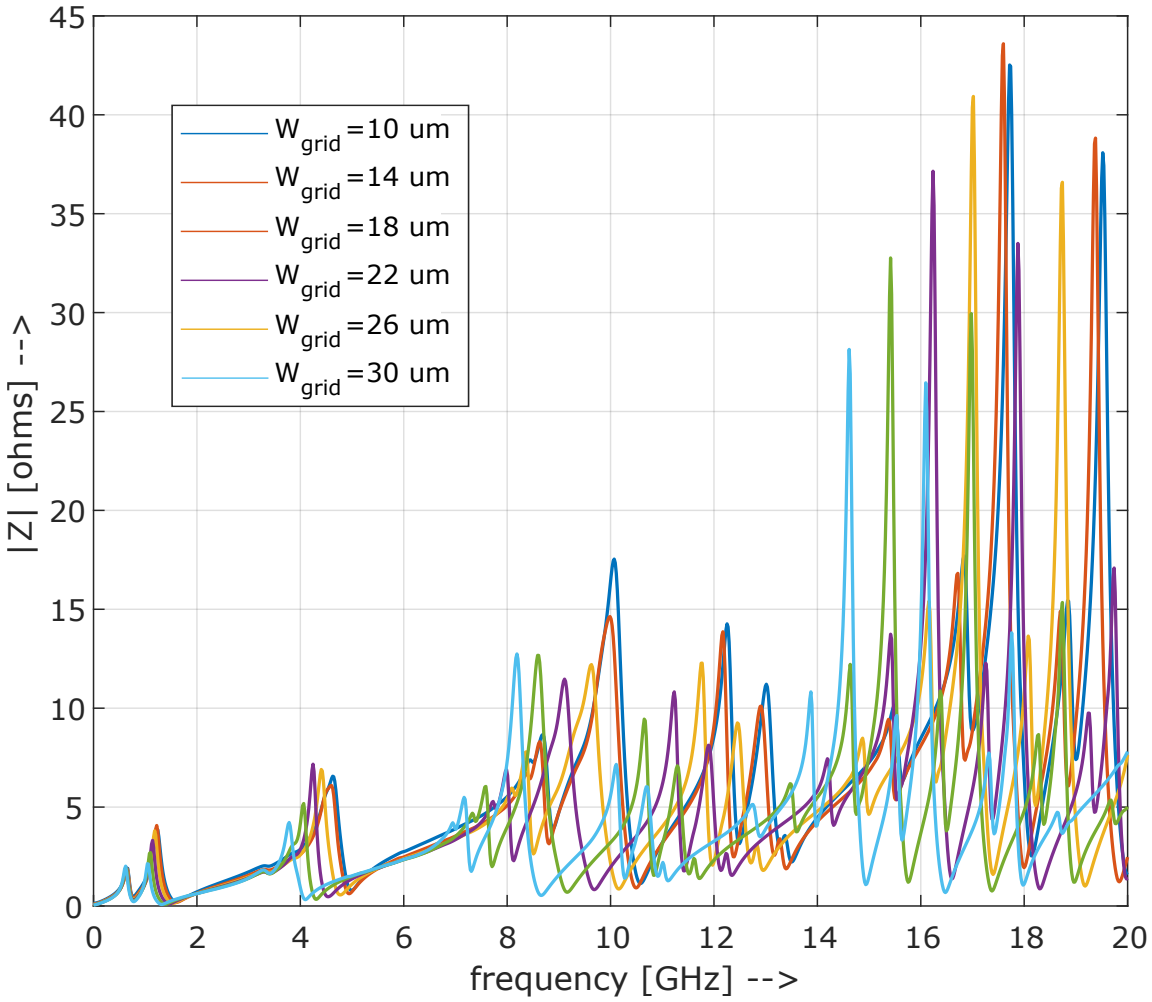


Figure 4.4: Impedance response changes with changing grid width

normalized root mean square loss function for frequency point k:

$$L_{real,k} = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (real(X_{i,k}) - real(y_i))^2} \tag{4.8}$$

$$L_{imag,k} = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (imag(X_{i,k}) - imag(y_i))^2} \tag{4.9}$$

where N is the batch size and **y** is the target complex-valued variable vector. The composite

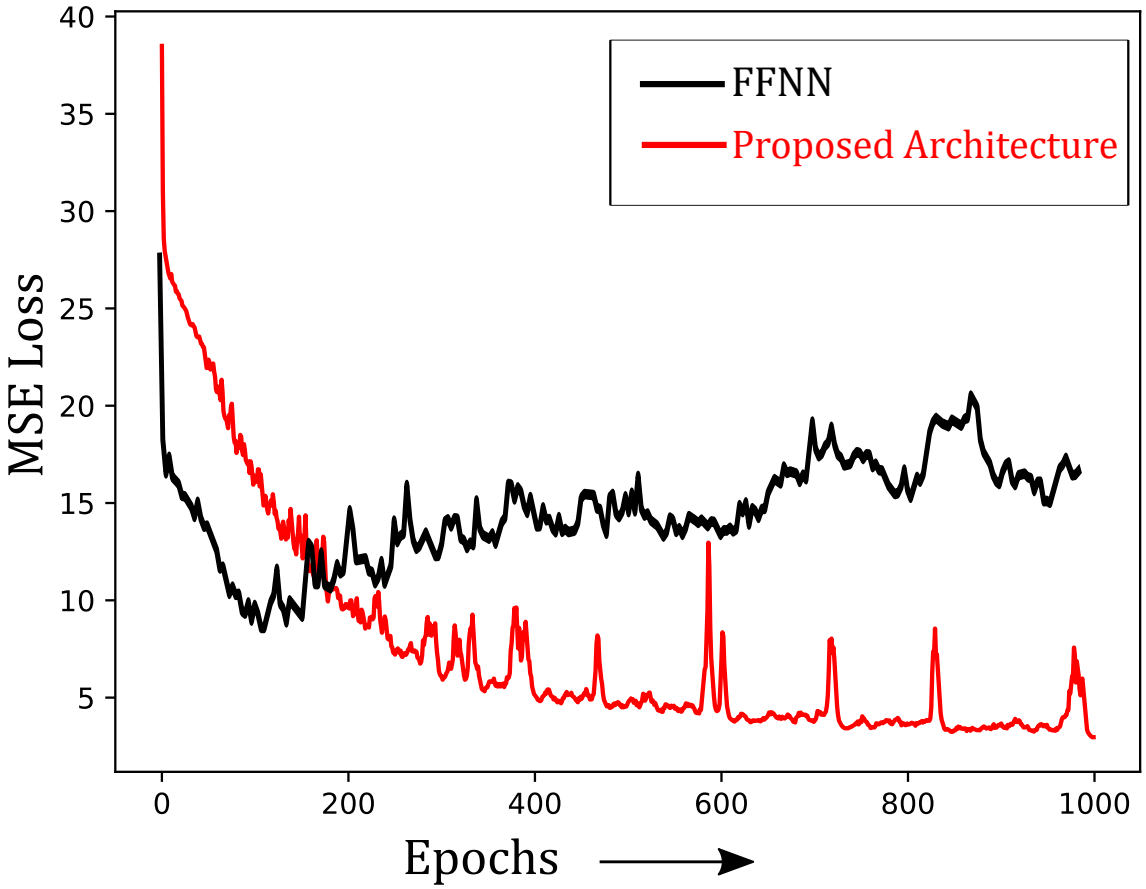


Figure 4.5: Loss Curves

loss is

$$L_k = \alpha * L_{real,k} + \beta * L_{imag,k} \tag{4.10}$$

where $\alpha, \beta \, \epsilon \, R$. After optimizing the values, we choose $\alpha = \beta = 1$. Finally, the loss for all frequency points is:

$$L = \frac{1}{F} \sum_{k=1}^{F} L_k \tag{4.11}$$

where F is the total number of frequency points. This loss is backpropagated to train the network.

We compare the results against a vanilla feed-forward neural network (FFNN) trained with the same loss function. The structure of the fully-connected neural network has been

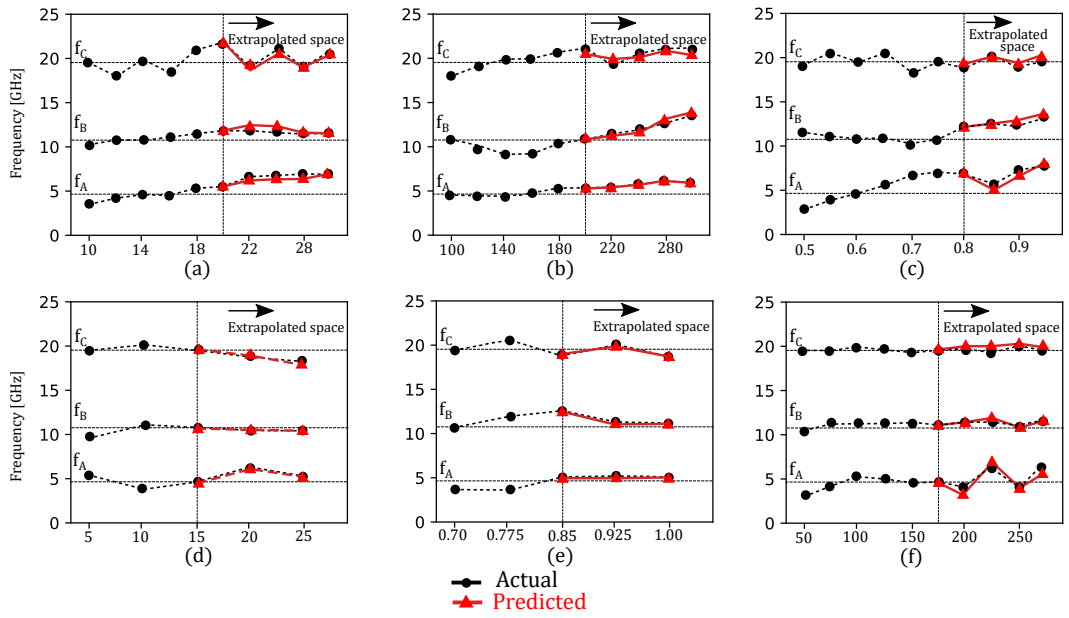


Figure 4.6: Extrapolation of individual design space parameters from a single trained model (a) Grid Width(μm), (b) Grid Spacing(μm), (c) Metal height(μm), (d) TSV radius (μm), (e) Substrate thickness (μm), (f) C4 radius (μm)

optimized to best fit the data. It contains 4 layers containing [10, 70, 200, 500] neurons respectively to generate 1000 frequency samples whereas our proposed architecture for this PDN comprises of 2 fully-connected layers and 3 transposed convolutional layers with $tanh(.)$ as the activation function used throughout. The model comparison is shown in Table Table 4.4. We can see that while our proposed architecture takes a longer time to train but the same loss value is minimized to 0.004 as opposed to 0.5 while running in a comparable time.

We evaluate the loss curves for both architectures in Figure 4.5. FFNN achieves convergence quicker than the proposed architecture but is stagnant thereafter.

Table 4.4: Comparison of different models

	FFNN	**Proposed architecture**
Validation NMSE	0.5	0.004
Training time	1.5hrs	2hrs
Run time	0.206 sec	0.890 sec

4.3.1 Pole Tracking in design space

A complex-valued frequency response can be expressed in terms of its in-band poles and residues along with proportional factor using vector-fitting[12] given in the following equation.

$$f(s) = \sum_{i}^{N} \frac{r_i}{s - p_i} + d * s + e \tag{4.12}$$

where $s = -\sigma + wj$ is the complex frequency point, p_i is the i th pole, r_i is the i th residue $\forall i = 1, 2, ..., N$, d is the proportional factor and e is constant. As we move across the design space, the pole frequency moves as shown in Figure 4.4. We investigate this effect in-depth across six most sensitive design parameters. We choose three distinct frequency points whose nominal training phase values are $f_A = 4.8GHz$, $f_B = 11GHz$ and $f_C = 18.5GHz$. In Figure 4.6, we illustrate the progression from training space of design parameters to extrapolated space. For example for Figure 4.6(a), we extrapolate the grid width (w_{grid}) from $20\mu m$ to $30\mu m$ whereas the network is trained with the values from $10\mu m$ to $20\mu m$. The changes in most design space parameters is somewhat linear but the distinction lies in the fact that a model trained with sample PDN responses is able to extrapolate in a multidimensional space with a reasonable accuracy.

To show that the model can extrapolate successfully in all dimensions, we present the predicted frequency response nominal point prediction is given in Figure 4.7. One can

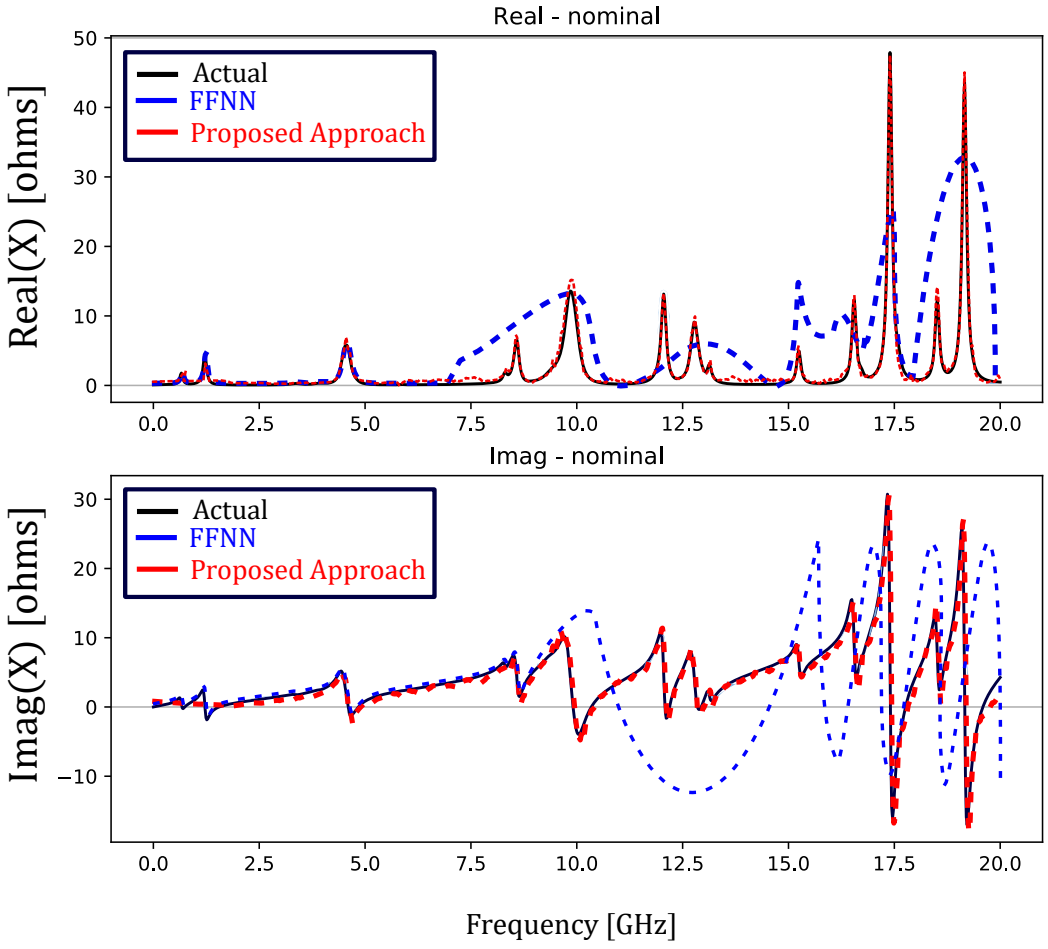


Figure 4.7: Frequency response of Nominal Value in the Extrapolation range, refer to Table Table 4.3

observe that our proposed network is able to identify nearly all the peaks shown till 20GHz. The prediction results is compared with the full-connected approach. At higher frequencies, the FFNN architecture fails to learn the frequency response because of a lack of learning data dependencies. This shows that convolutional architecture allows for parameter sharing and exploitation of spatial dependencies in data to learn a complex frequency response.

4.4 Timing Analysis

Application of ML techniques to design space and spectrum extrapolation have an inherent time and compute advantage that allows for assessing a large number of design variations with minimum cost. Given that the dimensions of each case are greater than five, a minimum of 3^5 design combinations is usually required to perform a thorough DSE using a three-point all-corner method. For this study, 100 design combinations are considered for the DSE. A summary of the comparison of simulation time for 100 design variations for three examples is shown in Table 4.5. This study is based on using a quad core i7 processor with no other workload running. The time taken to generate the system response using ML-based methods discussed above are compared to that of full-wave EM solver simulations, for each example, for the 100 different cases. The time shown for the EM solvers are the time required to simulate the number of design tuples. The benefit in compute time using ML models comes at a cost of minimal loss in prediction accuracy (as could be seen from the examples above), for a design or a set of frequency points outside the training range. We also show the test errors for each application. The test error is the normalized mean squared error over the entire frequency range over the test set.

4.5 Conclusion

We present a machine learning based approach to derive frequency response of distributed electromagnetic structures in general as a function of their geometrical and material properties. The model architecture consists of a fully connected upsampler which is a feed-

Table 4.5: Simulation time comparison for DSE with full-wave simulation and Machine Learning(ML) methods [**Dim** is dimensionality]

Test Case	**Dim**	**Freq. range**		**Freq**	**ML based method**			**EM solver**	**Test**
.	.	min	max	points	***Training***	***Inference***	***Total***	extrapolated tuples	**Errors**
Hairpin filter [50]	8	20 GHz	36 GHz	500	20 min	20 sec	23 min	1143 min	0.08dB
SIW filter [51]	11	110 GHz	170 GHz	850	25 min	25 sec	24 min	2067 min	0.95dB
Power Delivery [48]	14	DC	20 GHz	1000	25 min	33 sec	80 min	2872 min	1.13dB

forward neural network to produce the code in the high dimensional latent space. The code is fed to a transposed convolutional network to learn the frequency response at discrete frequency points. Results show that such an architecture performs better than only the fully-connected network approach and saves on computational time and resources in comparison to EM solvers. The proposed approach, while taking more time to train reduces the normalized mean squared error by more than 90% for power delivery applications.

CHAPTER 5

INVERSE DESIGN: RESPONSE SPACE TO DESIGN SPACE

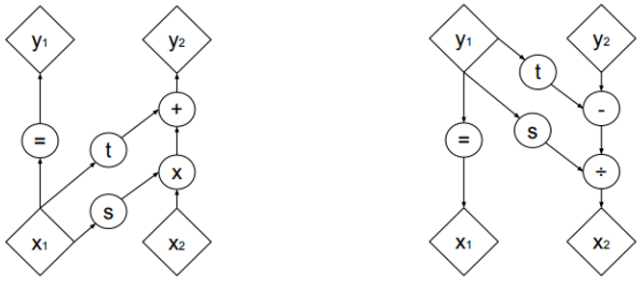
In this chapter, we describe the ongoing as well as the remaining work for the dissertation.

5.1 Problem Description

Recently, machine learning (ML) techniques have proven to be useful to form representations of data. A neural network maps inputs to outputs using a sequence of hidden layers. Mathematically,

$$y = h_L(...(h_2(h_1(x)))...) = f_\theta(x) \tag{5.1}$$

where h_i denotes the ith layer and x and y are inputs and outputs respectively. We can also define all the composition of the hidden layers in a single function f_θ where θ is a set of all parameters specifying the network. To solve the inverse problem $x = f_\theta^{-1}(y)$, there are two major challenges: (a) Existence of inverse function: $g_\theta = f_\theta^{-1}$ and (b) non-uniqueness with the likelihood that g_θ is not bijective. Using ML-based generative models, one can learn rich representation given design and response data which is then used to solve the inverse problem. Generative models for inverse design can be categorized into three categories: (i) Generative-Adversarial-Networks (GANs) [53] where two neural networks generator and discriminator are trained. However, GANs are trained using adversarial training schemes, which are difficult to implement and it does not solve the many-to-one problem, (ii) Variational Autoencoders (VAEs) [54] where encoder and decoder models are trained to learn the probability distribution of the latent space representing the output function. VAEs are trained to minimize the Evidence Lower Bound (ELBO) which trains the model parameters such as to best produce the given output. While these models are stochastic, they are approximately Bayesian and estimate, at best, the latent representation. This requires huge



(a) Forward propagation (b) Inverse propagation

Figure 5.1: RealNVP block enabling forward and backward propagation

samples of training data that can become a computational burden. (iii) Invertible neural networks (INNs) [55] can be used to learn distributions in data efficiently. In the ongoing work, we propose the use of INNs to find the suitable set of design parameters for power delivery networks given a set of desired envelope. Using INNs provides a rich insight into multi-variable space thus enabling the designers to work with multiple suitable design strategies.

5.2 Invertible Architectures

Invertible neural networks consists of invertible blocks designed by construction, enabled by bidirectional training and efficient sampling techniques.

5.2.1 Invertibility by construction

To make a model invertible, we build on top of realNVP [56] model. Considering an input variable X with D dimensions, it is randomly divided into two halves x_1 and x_2 each of $D/2$ dimension. It is passed through the forward block shown in Figure 7.3(a). The transformation becomes:

$$y_1 = x_1 \tag{5.2}$$

$$y_2 = x_2 * exp(s(x_1)) + t(x_1) \tag{5.3}$$

where y_1 and y_2 are output halves of dimension $D/2$ which are then concatenated through the same input shuffling order to make output Y. Here, $s(.)$ and $t(.)$ are any differentiable and monotonous transformations which means they can be approximated as fully connected neural networks. Computing the reverse path gives the block in Figure 7.3(b):

$$x_1 = y_1 \tag{5.4}$$

$$x_2 = \{y_2 - t(y_1)\} * exp(-s(y_1)) \tag{5.5}$$

Note that in the reverse path, we need not compute the inverse of the neural networks $s(.)$ and $t(.)$, hence there is no constraint on their invertibility. Hence, invertibility by construction is achieved.

5.2.2 Stochasticity

We obtain a stochastic machine learning model by change of variables technique:

$$p(y = f(x)) = p(x) * |J_{yx}|^{-1} \tag{5.6}$$

where $J_{yx} = \nabla_x(f)$ and f can be modeled as an invertible block from previous discussion.

5.2.3 INN Model

A composite INN architecture is shown in Figure 7.4. It is composed of invertible INN blocks where each block is followed by a shuffling block. The shuffling block shuffles the input data into halves for the next block. The depth of the network depends on total number of blocks num_{blocks} serves as a hyperparameter for the network. It depends on

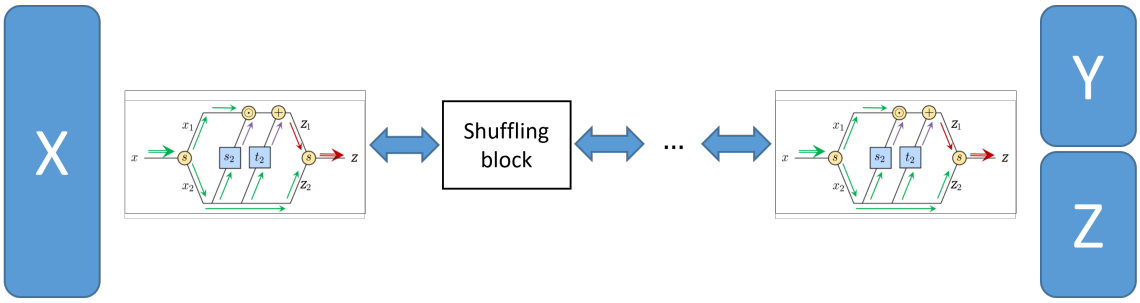


Figure 5.2: INN architecture

data complexity. Additionally, we need to make both sides dimensionally consistent. The constraint becomes:

$$D_{total} = D_x + D_{0_x} = D_y + D_z + D_{0_{yz}} \tag{5.7}$$

where D_x is the dimensions of x, D_y is the dimensions of y, D_z is the dimensions of z. We pad appropriately extra zeros D_{0_x} and $D_{0_{yz}}$. To ensure that the dimensions of the input and output are same, we pad the auxillary variable z alongwith the appropriate zero-padding. The augmented input and output vectors become $X_{aug} = [x; 0_x]$ and $Y_{aug} = [y; z; 0_{yz}]$ respectively.

5.2.4 Training

Once the INN model is set up, the network is trained. Consider the training dataset $TS = \{X_{TS_i}, Y_{TS_i}\}_{i=1}^{N}$ where N is the number of training samples. First, we go through the forward pass: $[y_{pred}; z_{pred}] = f_\theta(x)$ and measure two losses. The first is the mean-squared error (MSE) regression loss L_y between y_{pred} and y_{TS}:

$$L_y = \frac{1}{B}\sum_{i=1}^{B}(y_{pred,i} - y_{TS,i})^2 \tag{5.8}$$

where B is the batch-size in one epoch. The second is the probability divergence loss L_z between samples z_{pred} and a random sample $z \sim p(z)$. This loss ensures the independence

between the variables y and z. The loss is given by:

$$L_z = MMD(q(x, y), p(y)p(z)) \tag{5.9}$$

where p(z) is known, and $p(y)$ is given by eq. (Equation 7.11). $q(x, y)$ is a variational approximate joint distribution obtained by the forward pass and MMD(.) is the maximum mean discrepancy loss [**MMD˙paper**] which measures the similarity between two probability distributions. When y and z become independent, then the joint distribution will be closer to the product of the marginal distribution and $L_z \to 0$. Next, the reverse pass is run from the model: $x_{pred} = f_\theta^{-1}([y; z])$. We estimate a reconstruction loss L_x between x_{pred} and x_{TS} given as:

$$L_x = MMD(g(x_{pred}), p(x_{TS})) \tag{5.10}$$

where $g(x)$ is the variational distribution. When the reconstruction loss goes down, it means the reverse pass of the model is trained. All the losses combine to make the composite training loss:

$$L_{total} = w_x L_x + w_y L_y + w_z L_z \tag{5.11}$$

where w_x, w_y and w_z are the relative respective weights for the losses. The training algorithm pseudo-code is mentioned in Algorithm algorithm 3.

5.2.5 Inference

Once the training is done, we can perform inference on the model. For a given target output $\hat{Y}$, we sample z from $p(z)$ multiple times to compute the probability posterior $p(X|\hat{Y})$. This distribution is the inverse design desired. In multidimensional case, we get $p(X|\hat{Y})$. To find marginal distribution for a single variable x_i where $i = 1, 2, ..., D_x$, we apply the marginalization property:

$$p(x_i|\hat{Y}) = \int p(x_i, X_j|\hat{Y}) dX_j \tag{5.12}$$

Algorithm 3: INN training

Input: n_epochs, learning rate: $\alpha, p(z) = \mathcal{N}(0, \mathbb{I}_{D_z})$
Output: training loss, trained model
while $i \leq n\_epochs$ **do**
  for *x_batch, y_batch* $\in (X_{TS}, Y_{TS})$ **do**
    $[y_{pred}, z_{pred}] = f_\theta(x\_batch)$
    $L_y = MSE(y_{pred}, y\_batch)$
    $L_z = MMD(q(y, z), p(y)p(z))$
    sample $z \sim p(z)$
    $x_{pred} = f_\theta^{-1}([y\_batch, z])$
    $L_x = MMD(g(x), p(x))$
    $L_{total} = w_x L_x + w_y L_y + w_z L_z$
    $\forall p \in model.parameters() :$
    $p \leftarrow p - \alpha * grad(L_{total})$
  end
end

where the vector $X_j = [x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_{D_x}]$ contains all the dimensions except the one whose marginal distribution is to be estimated.

5.3 Numerical Example: Power Delivery

We can apply our approach to the power delivery example as well. Designers evaluate a PDN in terms of its impedance response Z_{PDN} as a function of the operating frequency. The impedance is desired to be less than a threshold value Z_{target} over a range of frequencies. A typical PDN is shown in Figure 5.3 which is adopted from [57]. It consists of voltage regulator module (VRM), power ground plane (P/G), C4 bumps, microbumps, through silicon vias (TSV) and decoupling capacitors.

Designers use CAD tools to extract the response to evaluate it for its reasonableness. At a given frequency f, the impedance is given as a function of the design space:

$$Z_{PDN} = T(X_{PDN}, f) \tag{5.13}$$

where $T(.)$ is the forward mapping transformation, and the design space X_{PDN} consists

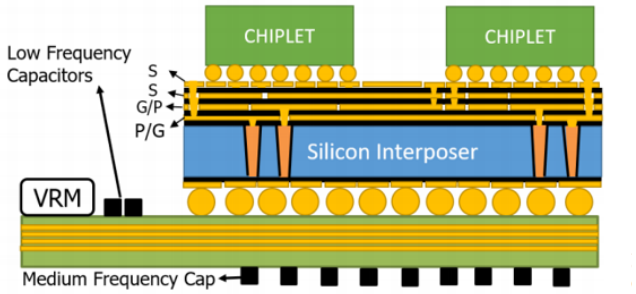


Figure 5.3: Illustration of a power delivery network [57]

of all the geometrical and material properties that characterize the response. We simulate the PDN using analytically derived formulae to compute the impedance response using analysis tools available online [58]. The PDN consists of (i) interposer P/G grid (ii) PCB P/G plane (iii) C4/μ-bump array (iv) TSV array and (v) via array. The impedance from each component is calculated and then put together to give the PDN impedance (Z_{PDN}). The design space is shown in Table Table 5.1 and the response is simulated from 1MHz to 20GHz with 1000 uniformly spaced frequency points.

Table 5.1: PDN characterization parameters

Parameters	**Symbol**	***Min value***	***Max value***
Conductivity	σ	1 x 10^7 S/m	8 x 10^7 S/m
Metal height	t_{metal}	0.5 μm	0.8 μm
Grid width	W_{grid}	10 μm	20 μm
Grid spacing	w_{grid}	100 μm	200 μm
TSV radius	r_{TSV}	5 μm	15 μm
TSV pitch	p_{TSV}	15 μm	60 μm
C4 radius	r_{C4}	50 μm	175 μm
C4 pitch	p_{C4}	0.15 mm	0.5 mm
μ-bump radius	r_{μ}	10 μm	12 μm
μ-bump pitch	p_U	40 μm	45 μm
Substrate thickness	h_{imd}	0.7 μm	0.85 μm
Si-dielectric	ϵ_{Si}	11.9 ϵ_o	
Poly-dielectric	ϵ_{poly}	3.9 ϵ_o	

Inspecting a template of the impedance response in log-scale shows three distinct regions. We have three frequency points f_1, f_2 and f_{max} as the maximum simulated frequency. We define the envelope of the impedance response by dividing it into three regions. Mathematically, the piece-wise function is:

$$E(f) = \begin{cases} R & \text{for } 0 \leq f \leq f_1 \\ L * (f - f_1)^2 + R & \text{for } f_1 \leq f \leq f_2 \\ B_s * f + B_c & \text{for } f_2 \leq f \leq f_{max} \end{cases} \tag{5.14}$$

where R is the constant coefficient before f_1, L is the quadratic coefficient from f_1 to f_2, and B_s, B_c are the linear slope and intercept coefficients from f_2 to f_{max}. This is illustrated in **??**. Hence, for the INN model, the input is a design matrix X of size $N\text{x}D_x$ and the output are the envelope coefficients $Y = [R, L, B_s, B_c, f_1, f_2]$ since f_{max} is same for all the examples.

For our INN model training, we have 800 training points and 200 testing points. We show the dimensions as well as the input/output characterization for the model in **??**. For our example, we use a 4-dimensional latent variable z whose distribution is assumed to be standard Gaussian. To verify our trained model, we show the forward results from the test set. In this case, we set the target index Y_{target} as indicated in Fig.**??**.

$$\begin{aligned} Y_{target} &= [R, L, B_s, B_c, f_1, f_2] \\ &= [0.048, 9.09, 1.55, 2.7, 21MHz, 560MHz] \end{aligned}$$

At the input side, we get a joint 7-dimensional posterior. We marginalize over all inputs and achieve the distributions in **??**. We sample the trained model 10^6 times. The orange bar indicates the values of the input tuple from the test set that generated the Y_{target}. We observe that in most of the cases, the reference value from the test set lies on top of the distribution where the model is most confident. This means the model has learnt the com-

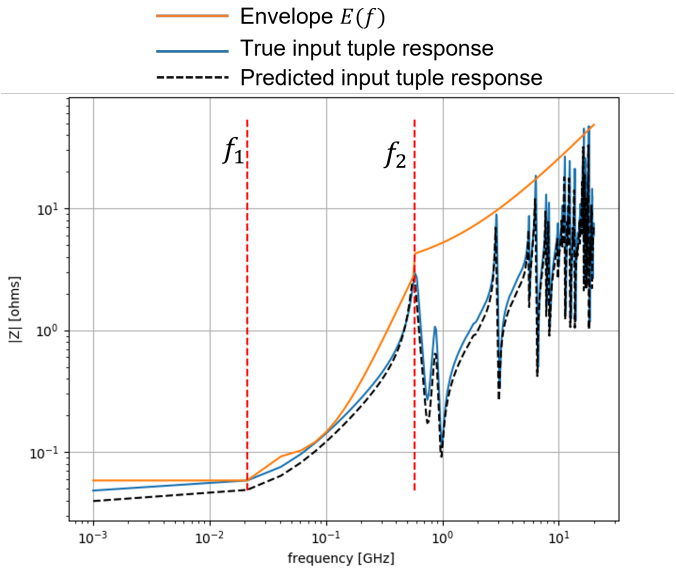


Figure 5.4: Comparison of impedance response for ground truth input tuple and predicted input tuple

plex 7-dimensional posterior distribution. Since, multiple design tuples might be producing close envelopes, we simulate the design tuple we get from the INN model where it is most confident i.e. the peak of the distribution and derive its envelope. The result of the process is shown in Figure 5.4. We see that, when we simulate the predicted input design tuple, we achieve an envelope that is very close to the target.

5.4 Conclusion

In this chapter, we harness flow-based invertible neural networks to model inverse mapping of the geometrical and material properties to the impedance envelope spec of a typical power delivery network. The proposed approach shows rich inverse distributions learnt by the model. We can quantify the uncertainty in our predictions by analyzing the variances of the posterior distributions of all the inputs.

CHAPTER 6

UNCERTAINTY QUANTIFICATION AND COMPARISON OF INVERTIBLE ARCHITECTURES

This chapter consists of two parts. In the first part, we perform a comparison of state-of-the-art inverse design strategies for a high-speed link. We take into account three machine learning architectures: (1) traditional fully-connected neural networks, (2) conditional generative adversarial networks and (3) invertible neural networks. The metrics for evaluation include quantitative mean-squared-errors on the test set as well as qualitative posterior distributions' similarity to the actual posterior distributions. We find that, on average, invertible neural networks have minimum mean-squared error for the input design tuple and their posterior shapes are in accordance with the actual distributions. Recently, machine learning (ML) techniques have been quite successful to find forward and inverse mappings between inputs and outputs [59], [60]. Neural networks minimize the error between the predictions and actual outputs by training using back-propagation. However, traditional neural networks are not invertible. Furthermore, generative models have been developed to output posterior conditional distributions instead of one deterministic design solution. Not only does this enable the designer to have multiple candidate choices but also give an evaluation of the reliability of the model. Hence, several generative models have been used for inverse design scenarios. In this chapter, we use a case study of a high-speed link for comparing three popular inverse design strategies: (1) traditional fully-connected neural networks (FCNN), (2) conditional generative adversarial networks (cGAN) [61] and (3) invertible neural networks (INN) [55] which have been used recently for channel design [62]. We perform the comparison both quantitatively and qualitatively. The inverse design flow is shown in Fig. Figure 6.1.

In the second part, we present a machine learning based tool to quantify uncertainty

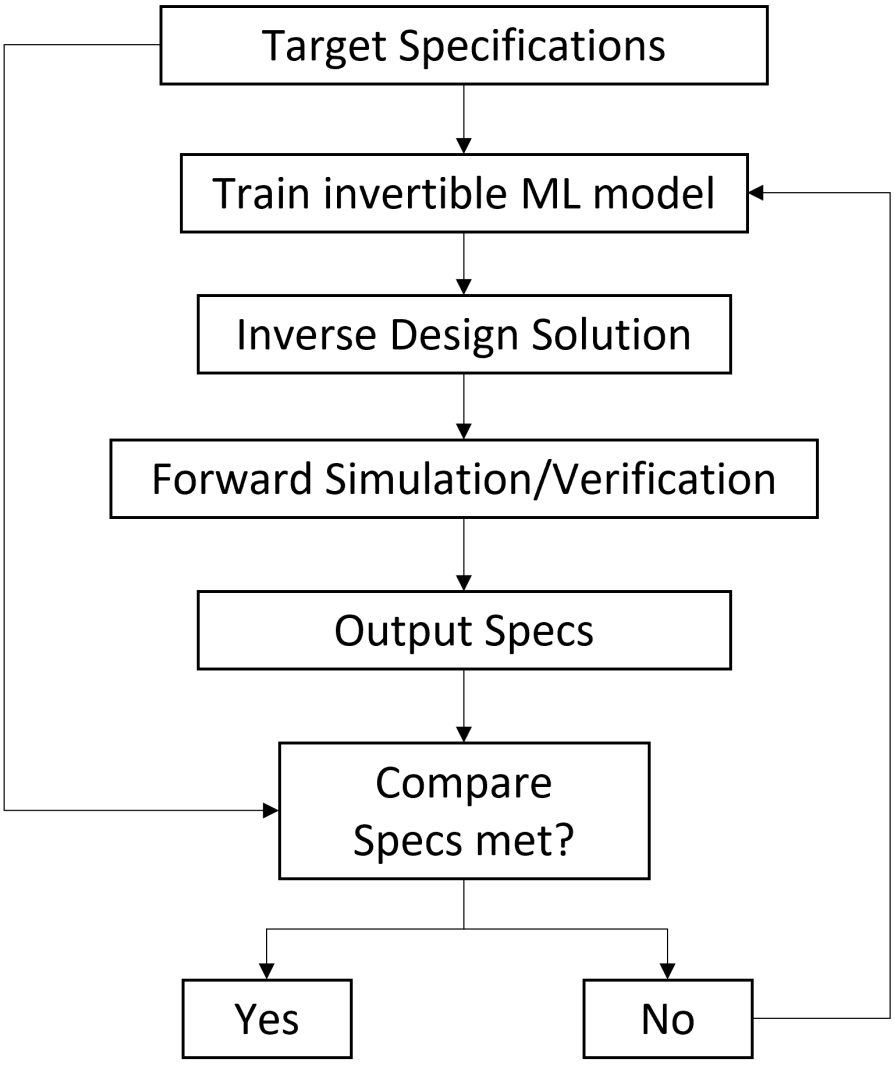


Figure 6.1: Inverse Design Flow

for prediction problems regarding signal integrity. Harnessing invertible neural networks, we convert the inverse posterior distribution given by the network to address uncertainty in frequency responses as a function of design space parameters. As an example, we consider a differential plated-through-hole via in package core and predict S-parameters from its geometrical properties. Results show 3.3% normalized mean squared error when compared with responses from a fullwave EM simulator.

6.1 Invertible Architectures

We compare three foundational invertible architectures as described below.

6.1.1 Fully Connected Neural Networks (FCNNs)

FCNNs form forward mapping relationships between input and outputs. They comprise of stacked layers of perceptrons which are a non-linear transformation of sum of scaled inputs. In the inverse problem, we can form a mapping between the design space x and the response space y as follows:

$$x = h_L(...(h_2(h_1(y)))...) = f_\theta(y) \tag{6.1}$$

where h_i denotes the ith layer and x and y are inputs and outputs respectively. We can also define all the composition of the hidden layers in a single function f_θ where θ is a set of all parameters specifying the network.The goal of training the FCNN is to minimize the comparison criterion between the predicted and actual design values. In order to not overfit the data, we make use of regularization techniques for better generalizability. An illustration of a traditional FCNN is shown in Fig. Figure 6.2.

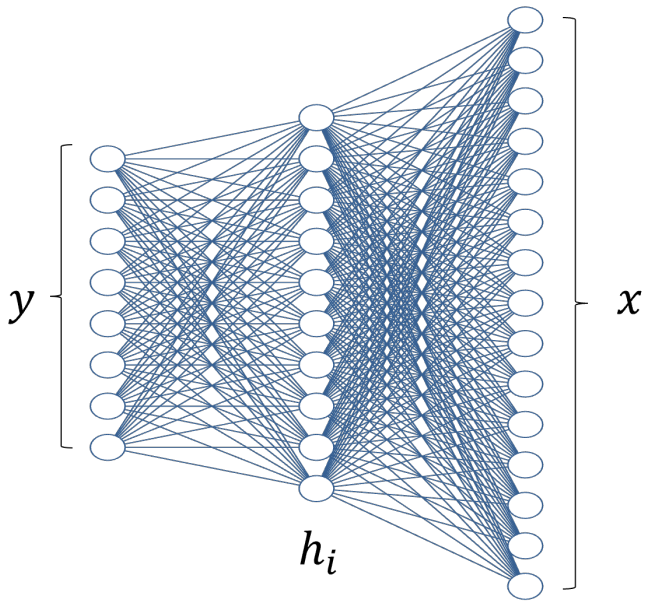


Figure 6.2: Traditional fully-connected neural network architecture (x: input, y: output, h_i: ith hidden layer, $i = 1, .., L$)

6.1.2 Conditional Generative Adversarial Networks (cGAN)

cGAN consists of modified adversarial generator and discriminator neural networks for conditional data generation. Instead of feeding the generator with latent noise, we stack the generator input with target labels desired. The output of the generator is the predicted inverse design solution. The discriminator takes two sets of tuple as inputs. The first set contains the real labels with real inverse solutions and compared with the cross-entropy loss. The second set of the discriminator inputs consists of fake predicted tuples generated by the model and real labels. Here the discriminator is trained against the adversarial loss. In essence, the generator and the discriminator play a mini-max game where the generator tends to mimic actual inverse design solution and the discriminator differentiates between the real and fake samples. The architecture for cGAN is shown in Fig. Figure 6.3.

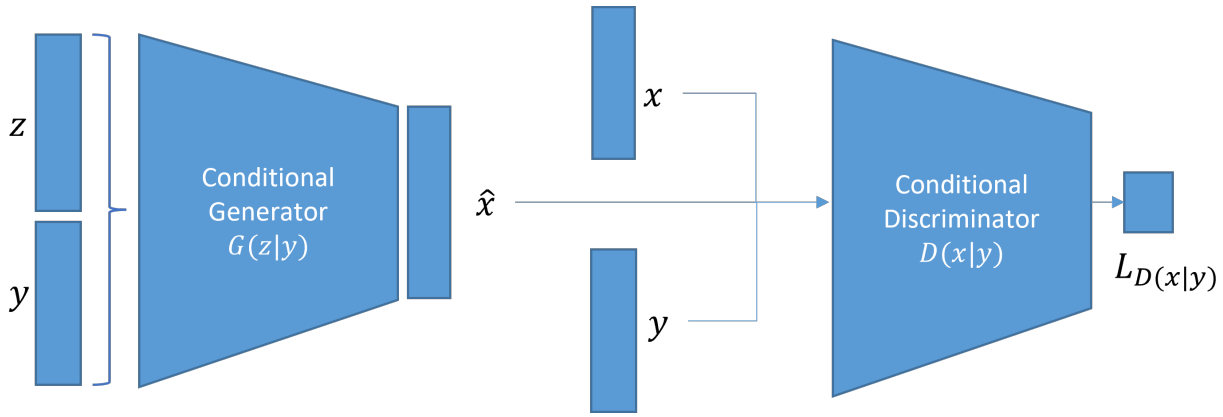


Figure 6.3: Architecture for cGAN (x: input, y: output, z: latent noise, $\hat{x}$: inverse design solution)

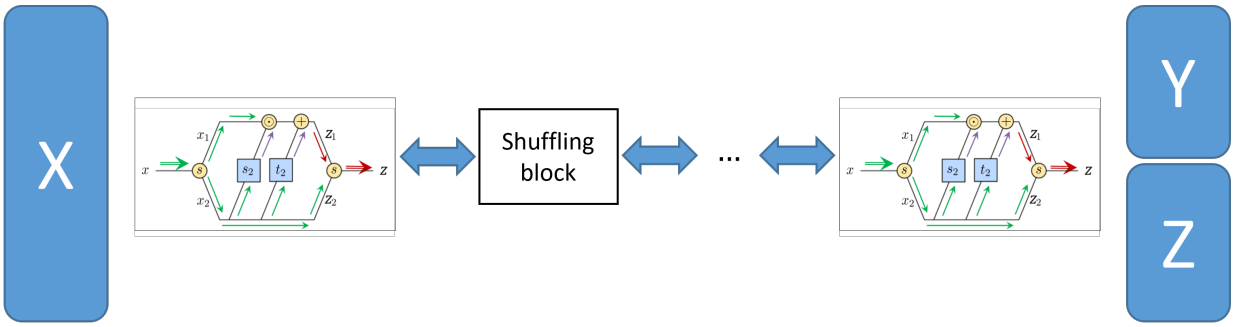


Figure 6.4: Architecture of Invertible Neural Network(x: input, y: output, z: latent variable)

6.1.3 Invertible Neural Networks (INN)

INN consists of inputs x and outputs y, linked by a set of invertible learning blocks interspersed with permutation and shuffling operations. On the output side, there are latent variables z added to encode the information loss that happens at the time of forward modeling. These latent variables are sampled from known distributions, which, when passed through the trained network in the reverse direction, conditioned on an output y, result in the conditional posterior distributions $p(x|y)$. The architecture is shown in Fig. Figure 6.4. The basic building block of the INN is an elegant arrangement of two complimentary affine coupling layers. These affine coupling blocks can be modeled as fully-connected neural networks that need not be invertible [56]. The transformation of the input $\boldsymbol{x}$ to output $\boldsymbol{y}$ is trivially reversible in-spite of having function coefficients e^{s_i} and t_i which are built of

complex, fully connected neural networks [62]. The inputs and outputs are made of equal dimensions by zero padding to retain symmetry of the structure and split into blocks that are transformed as shown in the equations. Due to the use of element-wise additive and multiplicative operations, the inverse of the transformation can easily be calculated without requiring the individual inverse of s_i and t_i, which could be intractable. The network update in the training phase happens after one forward-and-reverse pass.The training of the network is bi-directional. At every iteration, the gradients are updated after accumulating losses in the forward and reverse directions. The losses in the forward direction are (1) supervised loss, which in our case is the mean square error (MSE) between the simulated y and predicted y', (2) unsupervised loss on the joint distributions of the network outputs and the product of marginal distributions of the simulation outputs and known latent distributions, and (3) unsupervised loss on the distribution of the backward predictions on x and known prior distribution on x. Maximum mean discrepancy (MMD) is used to calculate the unsupervised loss, which only requires samples from the distribution. Compared to standard approaches, INN does not require a loss for direct posterior learning, which could be misleading in the absence of sufficient data capturing complex multi-modal distributions. The training of the INN is based on the forward process, which is known well.

6.2 Numerical Example: High-Speed Channel Link

High Speed Channel design is crucial to the successful operation of modern electronic packages and integrated systems. Primarily, it is desired that the channel achieves a suitable set of eye height and eye width specifications at the frequency of operation. However, to meet specifications for a high-speed channel, designers go through multiple iterations of trial and error which takes a lot of compute and design time. The input channel parameters such as length and width of the channel, the characteristic impedance and the equalization settings characterize the output eye parameters. Thus, the problem of inverse design of the channel is to find the best suitable set of input parameters that obtains the target eye height

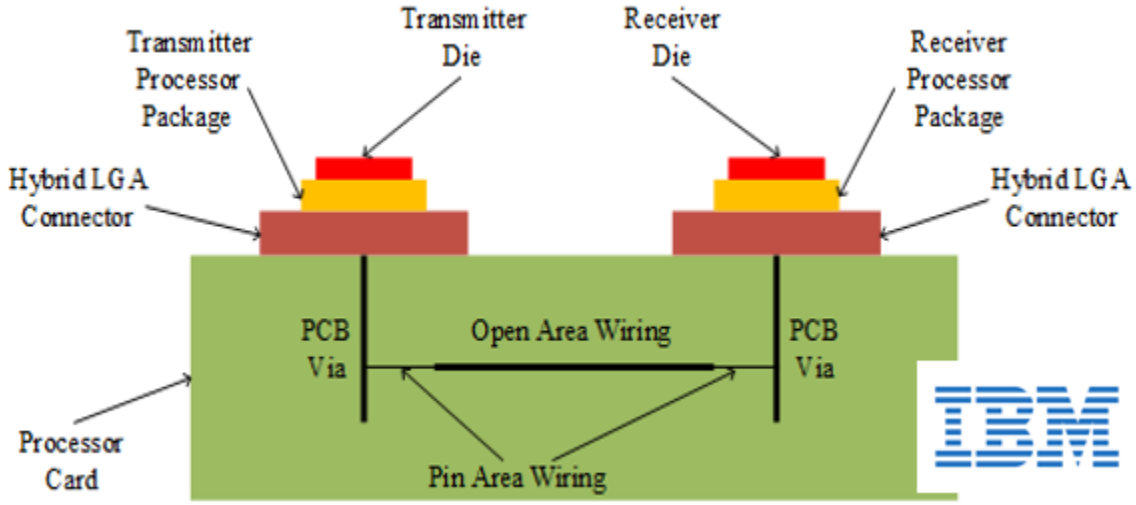


Figure 6.5: Commercial SerDes channel used in numerical example

and eye width at minimum. To this end, researchers have developed algorithms to optimize the design solution for a given channel spec.

6.2.1 Model Setup

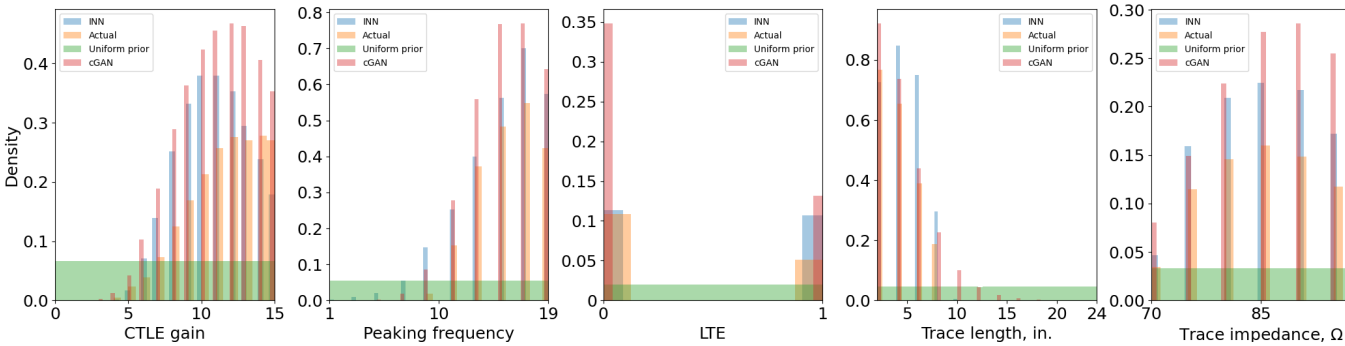


Figure 6.6: Posterior distributions for multiple models for the specified $y_{target} = \{EW = 85ps, EH = 110mV\}$

We consider a typical high-speed channel routed from the transmitter to receiver as shown in Fig. Figure 6.5. The design for signal integrity (SI) of such a channel requires (i) breaking it into a topology, and (ii) feeding the various models of the channel to a step response generator within the wrapper of a statistical eye analyzer. The equalizers are

placed at each end of this process and a random bit stream is passed through this system to generate an eye-diagram.

In this example, we use an IBM® XBus differential channel, operating at 16 Gbps, running between a transmitter and receiver sitting on land grid array (LGA) connectors. We explore a full-factorial channel simulation for all the variables in the design space, with 28860 channels simulated using IBM® internal solver called High-Speed SerDes Clock Data Recovery (HSSCDR). The design space consists of the length of the trace (L_{OA}) in the "Open Area Wiring" region shown in Fig. Figure 6.5 , the impedance of the trace (Z_{OA}) in that region and the receiver (Rx) equalizer design parameters. The equalizer used at the Rx is a continuous-time linear equalizer (CTLE) with the option of a long-tail equalizer (l). The design space limits and stepsparameters are shown in Table Table 6.1. The goal space is a set of four one-shot vectors. Each vector corresponds to a specification in eye-height (EH) and eye-width (EW). The joint-probability distribution of the EH-EW tuple is shown in Fig. Figure 6.7. The goal space was divided into four regions based on an arbitrary specification of EH $= 110$ mV and EW $= 85$ ps. Based on the values of EH and EW, the output can take one of four values of a four-bit one-hot vector from the set $\{1000$: if EH ≥ 110 mV and EW ≥ 85 ps; 0100: if EH < 110 mV and EW ≥ 85 ps; 0010: if EH ≥ 110 mV and EW < 85 ps; 0001: if EH < 110 mV, EW < 85 ps$\}$.

Table 6.1: CHANNEL DESIGN SPACE

Parameters	**Symbol**	***Min***	***Max***	**Step**
CTLE gain index	g_i	0	15	1
CTLE peaking frequency index	p_i	1	19	2
LTE state	l	0	1	1
Open area trace length	l_{OA}	2 in.	24 in.	2 in.
Open area trace impedance	Z_{OA}	70 Ω	100 Ω	5 Ω

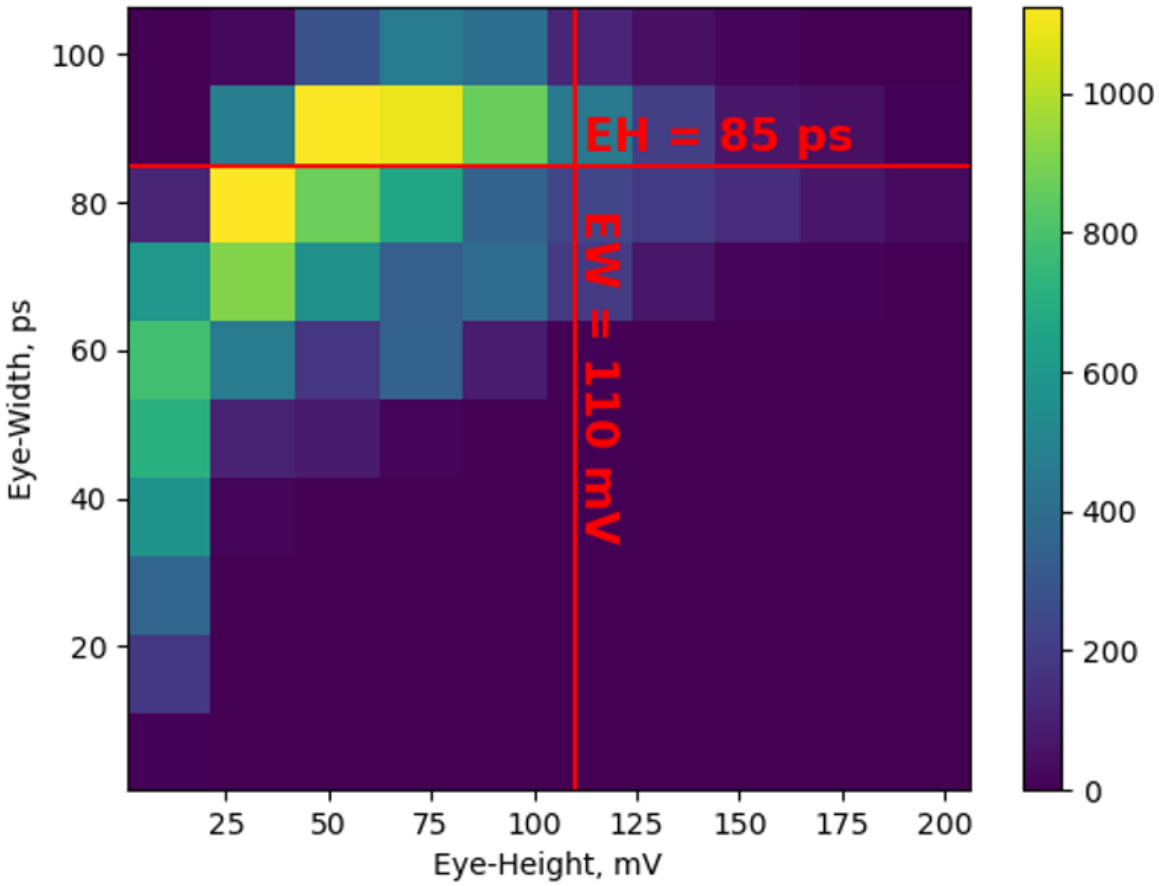


Figure 6.7: 2D Histogram of the EH-EW of full factorial channel design space

6.2.2 Results

To fully compare the approaches to inverse design, we present two perspectives: (1) Quantitative comparison, and (2) Qualitative comparison. For quantitative comparison, we pick the best thought solution by the model and compare it with the ground truth value in the test set. Furthermore, we compare the error produced by the predicted output and the actual target design point. We pick 3 tuples. Given y_{target}:

$$\hat{x}_{FCNN} = f_{FCNN}(y_{target}) \tag{6.2}$$

$$\hat{x}_{cGAN} =_x p_{cGAN}(x|y_{target}) \tag{6.3}$$

$$\hat{x}_{INN} =_x p_{INN}(x|y_{target}) \tag{6.4}$$

Since, the inverse mapping might not be one-to-one, it is possible that multiple design solutions achieve the same target spec. Hence, to be fair in our model comparison, we simulate the inverse design solutions obtained by all the models and compare the resulting eye parameters instead of comparing the input design tuples. In Table Table 6.2, we show the mean squared error of eye height and eye width for all the 2474 test set channels. Next, for qualitative comparison, we compare the posterior distributions of the generative models. As a sample test case, we consider the output spec: $y_{target} = \{EW = 85ps, EH = 110mV\}$. In Fig. Figure 6.6, we overlay the posterior distributions p_{INN} and p_{cGAN}. For our channel, we also calculated the exact actual posterior distribution p_{actual}. From the posterior distributions, we can see that p_{INN} and p_{cGAN} are quite similar but differ in the training and inference time. This is because of the difference in model complexity and size. Both models were trained and evaluated on a windows machine with 16GB RAM and processor Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz. We are working with

Python3.7.6 and Cuda toolkit v11.3.1. The GPU used was NVIDIA GeForce GTX 1050 with Max-Q Design.

Table 6.2: Model Comparison

Metrics	**FCNN**	**cGAN**	**INN**
$RMSE(EH_{true}, \hat{EH}_{model})$	7.60 mV	4.33 mV	2.45mV
$RMSE(EW_{true}, \hat{EW}_{model})$	10.11 ps	5.35 ps	2.22 ps
Training time	13 min	35 min	11 min
Inference time	35 ms	90 ms	11 ms
Model Size(parameters)	≈112k	≈ 547k	≈ 12k
Uniqueness information	No	Yes	Yes

6.3 Uncertainty Quantification

In this section, we propose to utilize INNs to achieve inverse posterior distribution $p(X|Y)$. We then pick the most probable sample points from this distribution and undergo a forward pass through the INN. This gives us the uncertainty bounds for the frequency response Y. This approach is illustrated in Fig. Figure 6.8. As an example, we consider a differential via pair and parameterize its design space. The S-parameters of the resultant structure are the output response.

After training the INN, inference is performed. We sample z coming from a known distribution $p(z)$, generally assumed to be a standard Gaussian. We append the sampled z with the target y_{target} and go through backward pass of the network to obtain the inverse posterior distribution $p(x|y_{target})$. The expected value of this distribution gives us the mean, and the variance of the distribution shows the sharpness of the input design tuple. The goal, here, is to obtain the variance and mean of the forward posterior distribution $p(y|x)$. To achieve this, we consider a range of the most probable input design tuples provided by INN

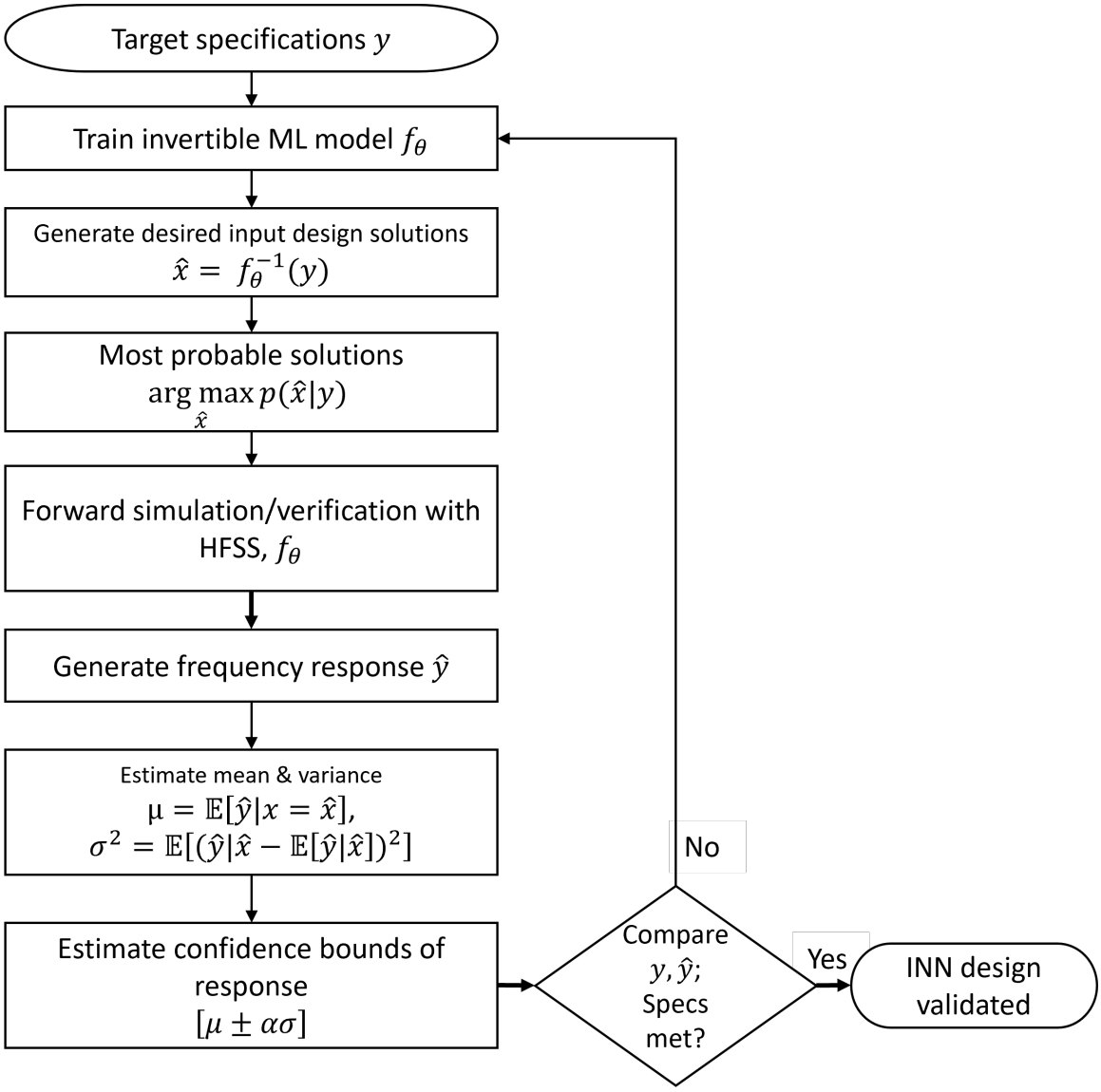


Figure 6.8: Flow of inverse design and its uncertainty quantification.

Table 6.3: Control Parameters of the PTH Structure

Parameter		Unit	Min	Max
μ-via Diameter	$d_{\mu\text{-}via}$	μm	30	70
μ-via Pad Diameter	$d_{pad,\mu\text{-}via}$	μm	31	140
BU Layer Thickness	h_{BU}	μm	20	35
μ-via Top Antipad Radius	$r_{a,BU,TOP}$	μm	100	500
μ-via Bot. Antipad Radius	$r_{a,BU,BOT}$	μm	100	500
PTH Pitch	v_P	μm	300	1200
Core Thickness	h_{core}	μm	100	1200
BU Cu Thickness	$t_{c,BU}$	μm	10	20
Core Cu Thickness	$t_{c,Core}$	μm	11	40
PTH Diameter	d_{PTH}	μm	100	250
PTH Pad Diameter	$d_{pad,PTH}$	μm	110	500
PTH Top Antipad Radius	$r_{a,PTH,TOP}$	μm	50	500
PTH Bot. Antipad Radius	$r_{a,PTH,BOT}$	μm	50	500

and undergo a forward pass to obtain upper and lower confidence bounds for our frequency responses.

6.4 Numerical Example: Differential PTH Pair in Package Core

We consider an application of modeling a differential plated-through-hole (PTH) pair in package core along with the microvias that connect to build-up layers. Such structures are common since they enable vertical interconnection for signals. As such, the signal integrity of such differential vias is crucial for high-speed interfaces. We achieve an inverse surrogate model for the structure shown in Fig. Figure 7.11. Obtaining the inverse posterior enables us to quantify uncertainty in the S-parameters of the shown structure.

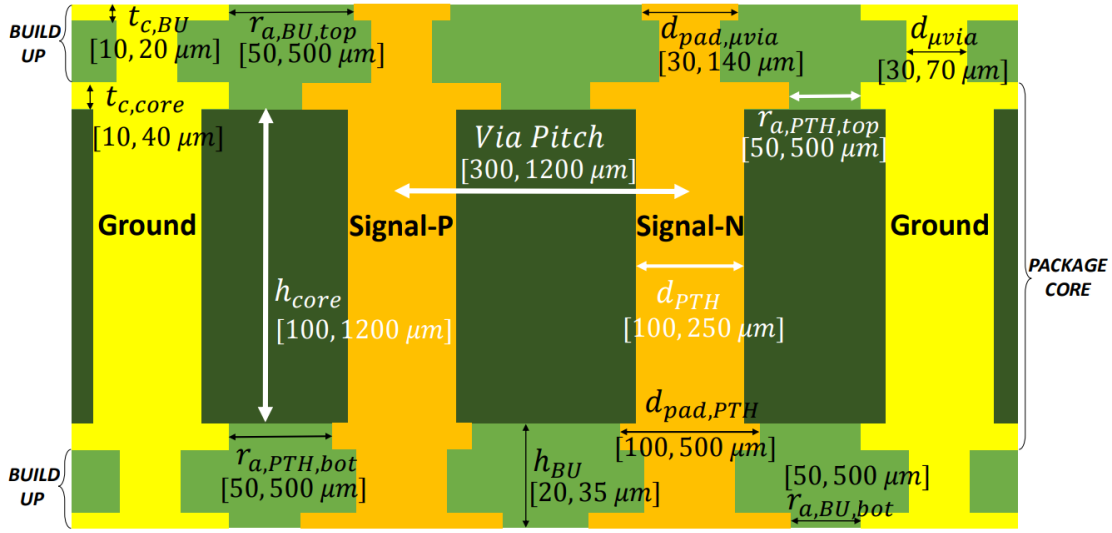


Figure 6.9: Parameters of the differential PTH in package core [63].

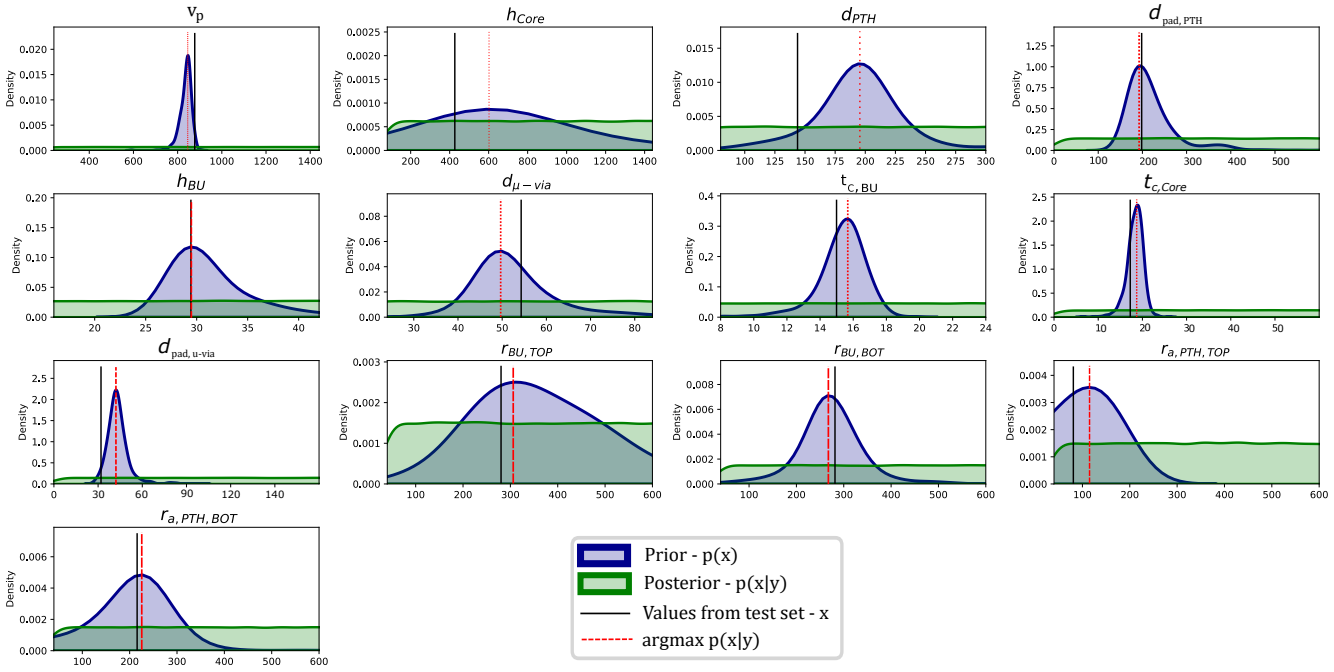


Figure 6.10: Inverse posterior distributions $p(\mathbf{x}|y_{target})$, black vertical line shows values from the test set

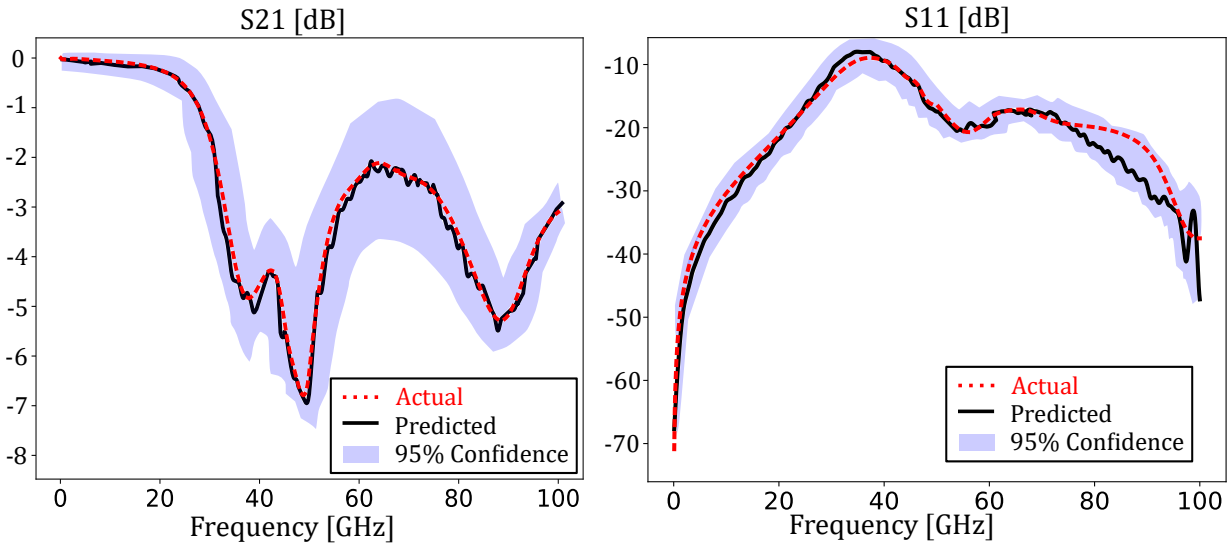


Figure 6.11: Forward simulation results comparison for INN predictions with 3D EM solvers

6.4.1 Model Setup

The design space is parameterized as a 13-D input design tuple. The minimum and maximum values are shown in Table Table 7.4. Each input combination in the design space has a corresponding four-port scattering (S) parameter matrix from 0.1-100 GHz with steps of 100 MHz. The objective is to determine an invertible mapping from the design space X and frequency response Y. Since the structure is partially reciprocal and symmetric, we only consider $S_{11}, S_{12}, S_{13}, S_{14}, S_{33}$ and S_{34}. We take the magnitude of the S-parameters, resulting in an output dimension of 6000. We draw 682 samples using Latin Hypercube Sampling (LHS) and obtain S-parameters using Ansys HFSS. The data is split into train and test sets for the INN model. We use 500 samples for training and the remaining for evaluation of the model.

6.4.2 Results

We train the INN for 50 epochs with 100 iterations per epoch optimizing the model with an intial learning rate of $\alpha = 0.01$ using Adam optimizer. We train with an adaptive

exponentially decreasing learning rate until the model converges. On random, we choose a desired response y_{target} from the test set. Next, we sample $z \sim p(z)$ for 5,000 times to obtain $x = f_\theta^{-1}(y, z)$. For this application, we choose the dimensionality of z to be 1000. The inverse distributions for each dimension of x is shown in Fig. Figure 7.12. We also plot the prior distributions before conditioning on y_{target}. Starting with a uniform prior, we see that the posterior inverse distribution becomes dense around a certain range of design tuples that the model suggests are most likely to produce the target distribution. For each dimension, we choose the design tuple for which the model is most confident. These input combinations are fed back into the INN to obtain a set of frequency responses. This range determines the lower and upper bounds for the target frequency responses.

We simulate the chosen input ranges into a forward simulator to obtain confidence intervals. In Fig. Figure 7.13, we plot the y_{target} from the test set coming from the 3D EM solver. We compare it with the output from the INN. We find that the mean of the predicted frequency responses from the INN closely matches the test set values. Specifically, we use the normalized mean-squared error as loss metric over each frequency response in the test set:

$$NMSE = \frac{1}{N_d D_y} \times \sum_{d=1}^{D_y} \sum_{n=1}^{N_d} \times \left(\frac{\sum_{m=1}^{N} (S_{n,d}[m] - \hat{S_{n,d}}[m])^2}{\sum_{m=1}^{N} (S_{n,d}[m] - \frac{1}{N} \sum_{m=1}^{N} \hat{S}_{n,d}[m])^2} \right) \quad (6.5)$$

where N_d are the number of evaluation designs for the model and $D_y = 6$ represents the magnitude of the learnt S-parameters. The NMSE value for the proposed approach is 3.3%.

6.5 Conclusion

In this work, we investigate the inverse design solutions for a high speed channel for 3 architectures: fully-connected neural networks, conditional generative adversarial networks and

invertible neural networks. We perform a deterministic as well as stochastic comparison. The quality of the posteriors generated by cGAN and INN are similar to actual posteriors. We conclude that inverse design, in general, can aid the designers to find a suitable input combination. Furthermore, we propose a method to perform uncertainty quantification of frequency response as a function of design space parameters using invertible neural networks for signal integrity applications. Specifically, we illustrate a differential plated-through-hole pair in package core as an example. We provide lower and upper confidence bounds for output 4-port S-parameters. We achieve a normalized mean-squared error of 3.3% on the test set.

CHAPTER 7

AINN: ADVERSARIAL INVERTIBLE NEURAL NETWORKS

In this chapter, we present Adversarial Invertible Neural Networks [64] - a machine learning based inverse design technique that generates the most suitable design solution for a desired response for microwave and electronic system applications. We harness flow-based invertible neural networks that are trained adversarially to make possible the inverse design of high-dimensional parameterized structures. We illustrate our technique on 3 examples: (1) a simple patch antenna, (2) Substrate Integrated RF waveguide and (3) a differential via pair in package. We compare our approach with other state-of-the-art inverse design techniques as well. Results show that the proposed approach is able to find more accurate solutions in comparison to its competitors. The posterior generated by the model depicts the uncertainty associated with the solution. We get 0.08dB, 1.1dB mean-squared error and 1.11% normalized mean error in the respective three examples.

7.1 Problem Statement

Modern electronic systems rely on efficient design for their successful operation. For microwave components, the response surface depends on the geometrical and material properties of the electromagnetic structures. Recently, the number of physical and geometrical dimensions has increased thus a more intricate design methodology is required. The total design flow for microwave modeling includes forward design and inverse design. Generally, forward design comprises of determining a system's forward response whereas the problem of inverse design is to find out the best set of design space combinations that would most likely lead to the desired response. For a design space X and observed noisy response space Y, one can write:

$$Y = T(X) + \epsilon \tag{7.1}$$

where ϵ is random noise usually modeled as a standard gaussian and $T(.)$ is the forward transformation. The goal of inverse design is to find the design space from the noisy observations:

$$X = T^{-1}(Y) \tag{7.2}$$

However, the challenges with this formulation are twofold: (a) the inverse transformation $G = T^{-1}$ might not exist and (b) the inverse transformation may not be unique which means it is a many-to-one problem. More traditionally, we can formulate the problem as an optimization program [**optimization˙PSO**]. The goal is to find the optimum design tuple X^* that minimizes the error in a regularized manner. Mathematically,

$$X^* = \min_X L(T(X), Y) + \lambda R(X) \tag{7.3}$$

where $R(.)$ is a regularizer and $L(.,.)$ is an error metric for minimization. Generally, traditional optimization programs might not converge to a solution leading to the existence problem identified earlier. Moreover, after converging, they output a single deterministic solution which does not have any information about other inverse solutions possible. To model the confidence in the solutions, probabilistic frameworks have been developed. The Bayesian technique investigates the problem in a stochastic manner:

$$p(X|Y) = \frac{p(Y|X)p(X)}{\int_X p(Y|x)p(x)dx} \tag{7.4}$$

where $p(X)$ is the prior that is generally available with the training data, the likelihood $p(Y|X)$ is found through the known forward transformation $T(.)$ and $p(X|Y)$ is the inverse posterior probability distribution to be estimated. While equation (Equation 7.4) while provides exact inverse posterior - it is analytically intractable and computationally expensive to compute. To bypass this computation, many machine learning(ML) approaches have been developed to directly estimate the distribution $p(x|y)$ using deep generative modeling.

One way to model the relationship between two variables is through feed-forward neural networks (FFNN). Such a structure maps the input function space to the output function space. Given the dataset $D = \{x_i, y_i\}_{i=1}^{N}$, FFNN can be represented as stacked intermediate non-linear layers to generate the output. To achieve the inverse solution, we input the observations y and output the solution x to model the inverse function T^{-1}:

$$x = h_L(...(h_2(h_1(y)))...) = f_\theta(y) \tag{7.5}$$

where h_i denotes the ith layer and x and y are inputs and outputs respectively. We can also define all the composition of the hidden layers in a single function f_θ where θ is a set of all parameters specifying the network. The goal of training the FFNN is to minimize the comparison criterion between the predicted and actual design values. Recently, there has been much work in microwave domain using FFNNs [65]. However, the feed-forward network has no guarantees about uniqueness and reliability of the solution.

More recently, deep generative models have been developed to model input-output distributions using a latent variable z. There are three main architectures: (a) Variational Autoencoders (VAEs) [54], [66] where encoder and decoder models are trained to learn the probability distribution of the latent space representing the output function. VAEs are trained to minimize the Evidence Lower Bound (ELBO) which trains the model parameters such as to best produce the given output. While these models are stochastic, they are approximately Bayesian and estimate, at best, the latent representation. This can require huge samples of training data that can become a computational burden, (b) Generative adversarial networks, [53] [67], where two neural networks generator and discriminator are trained. While these models can work efficiently, they require a lot of training data and are deterministic in nature and it does not solve the many-to-one problem and (c) Invertible Neural Networks (INN) are flow-based generative models that consists of stacked invertible transformations and can find the exact tractable posterior at inference time. A flow-chart

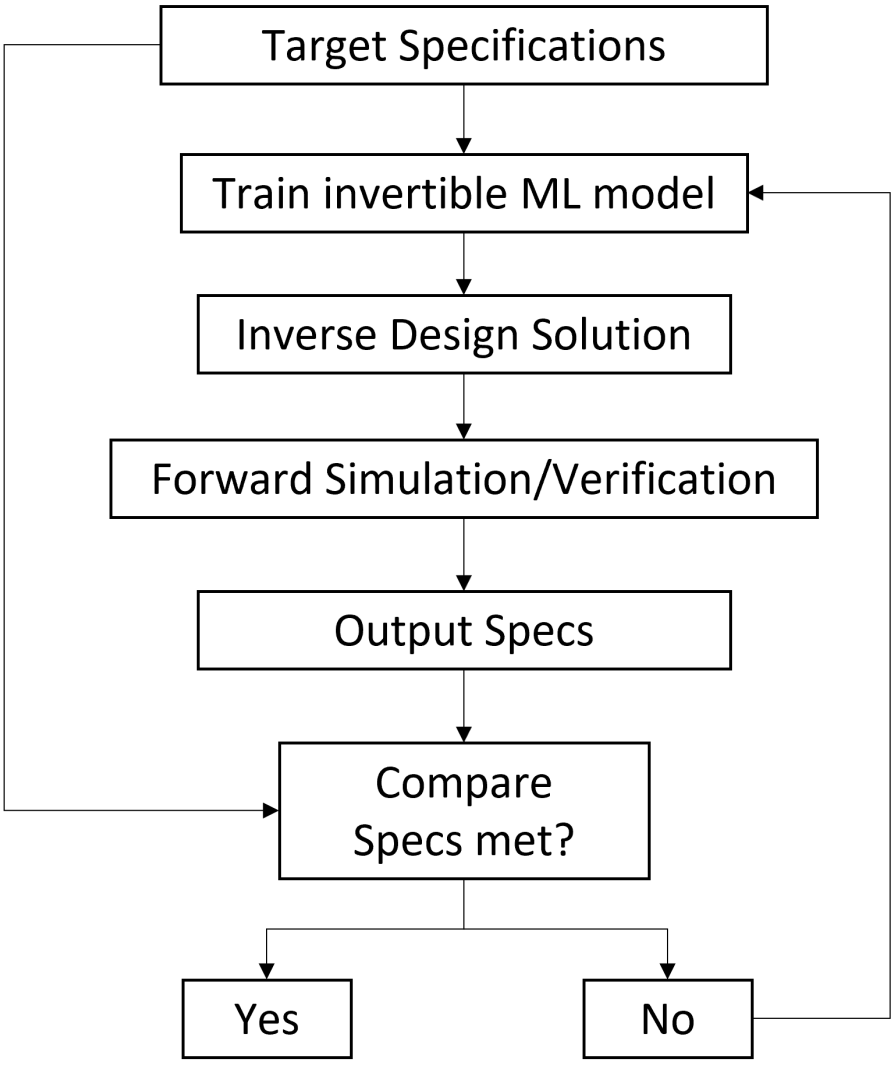


Figure 7.1: Inverse Design Flow [69]

for inverse design is illustrated in Figure 7.1. While training INN uses maximum mean discrepancy (MMD), [68], as a measure of dissimilarity between two probability distributions using samples. The metric is defined as the maximum difference in means of distributions when transformed into a hilbert space. Mathematically,

$$MMD(x, y) = \sup_{f \in F} ||\mu_x - \mu_y|| \tag{7.6}$$

While MMD is a popular technique and has established statistical bounds [70], it has been proved that the metric breaks at higher dimensions. In this work, we propose a solu-

tion to harnessing the power of invertible neural networks by training them adversarially thereby bypassing the need for any error metric targeting high dimensional applications. Since neural networks work with scalable dimensions - we propose using discriminators to find if the sample generated by the model is coming from the predicted model or from the train set. More specifically, we can summarize our contributions as follows:

- For the first time in author's knowledge, we treat INN as a conditional generator instead of a whole system and train them using adversarial discriminators at the input and output side. We call this architecture as Adversarial invertible neural network.
- This gives rise to a hybrid generative model by combining the expressiveness of INNs and the adversarial training schemes of GANs
- We harness INN as stochastic generative model to quantify uncertainty that the inverse design solution gives with regards to the desired parameters.

Much like GANs, the INN and the discriminators play a minmax game where the goal of the INN is trained to maximized its likelihood of parameters given the data and produces fake inverse samples and the discriminators at both sides are trained to distinguish between fake and real samples. Furthermore, to validate our proposed architecture, we apply it in three scenarios: (a) a patch antenna, (b) SIW interconnect and (c) a differential via pair in package, given their importance in recent research areas.

7.2 AINN Architecture

In this section, we describe the architecture of AINN. It consists of an INN that acts as a generator and two discriminator neural networks that acts as an adversary to train the INN. The architecture is shown in Figure 7.2. The details are in the following subsections.

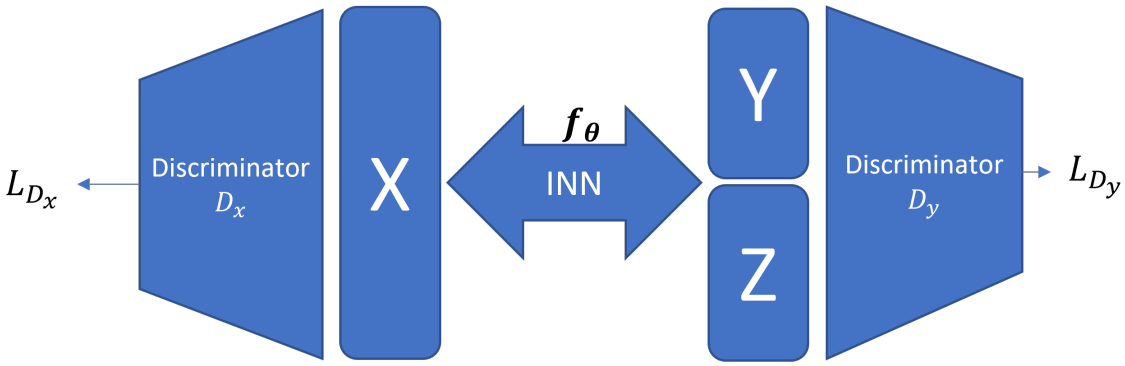


Figure 7.2: AINN architecture

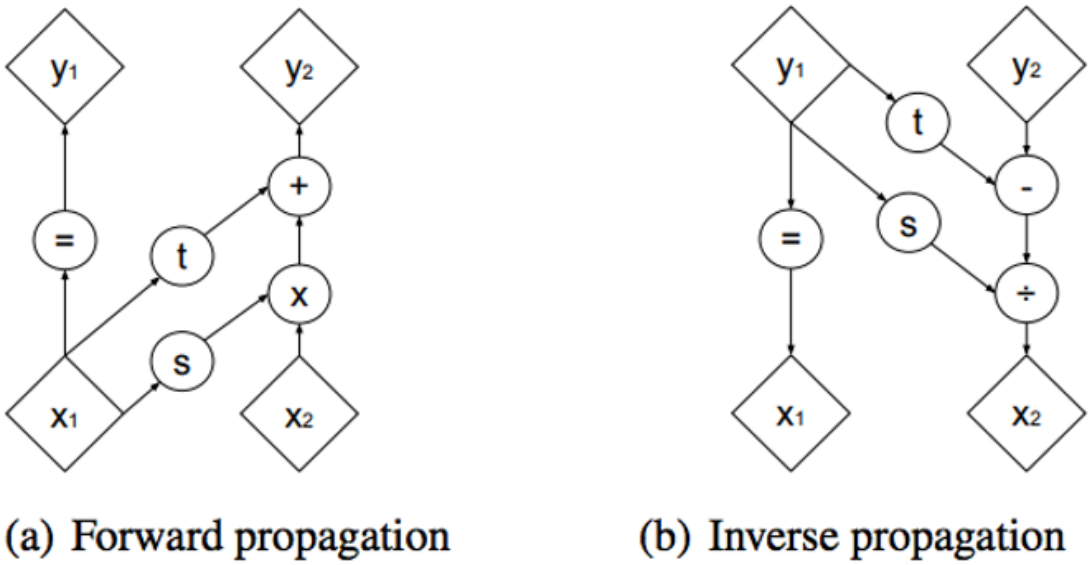


Figure 7.3: RealNVP block enabling forward and backward propagation

7.2.1 Invertible Architecture

Invertible neural networks [55] consists of invertible blocks designed by construction, enabled by bidirectional training and efficient sampling techniques. To make a model invertible, we build on top of realNVP [56] model. Considering an input variable X with D dimensions, it is randomly divided into two halves x_1 and x_2 each of $D/2$ dimension. It is passed through the forward block shown in Figure 7.3(a). The transformation becomes:

$$y_1 = x_1 \tag{7.7}$$

$$y_2 = x_2 * exp(s(x_1)) + t(x_1) \tag{7.8}$$

where y_1 and y_2 are output halves of dimension $D/2$ which are then concatenated through the same input shuffling order to make output Y. Here, $s(.)$ and $t(.)$ are any differentiable and monotonous transformations which means they can be approximated as fully connected neural networks. Computing the reverse path gives the block in Figure 7.3(b):

$$x_1 = y_1 \tag{7.9}$$

$$x_2 = \{y_2 - t(y_1)\} * exp(-s(y_1)) \tag{7.10}$$

Note that in the reverse path, we need not compute the inverse of the neural networks $s(.)$ and $t(.)$. Hence, invertibility by construction is achieved.

Since, we model the data samples coming from a distribution, a stochastic analysis is needed. We obtain a stochastic machine learning model by change of variables technique [71]:

$$p(y = f(x)) = p(x) * |J_{yx}|^{-1} \tag{7.11}$$

where $J_{yx} = \nabla_x(f)$ and f can be modeled as an invertible block from previous discussion. A composite INN architecture is shown in Figure 7.4. It is composed of invertible INN blocks where each block is followed by a shuffling block. The shuffling block shuffles the input data into halves for the next block. The depth of the network depends on total number of blocks num_{blocks} serves as a hyperparameter for the network. It depends on data complexity. Additionally, we need to make both sides dimensionally consistent. The constraint becomes:

$$D_{total} = D_x + D_{0_x} = D_y + D_z + D_{0_{yz}} \tag{7.12}$$

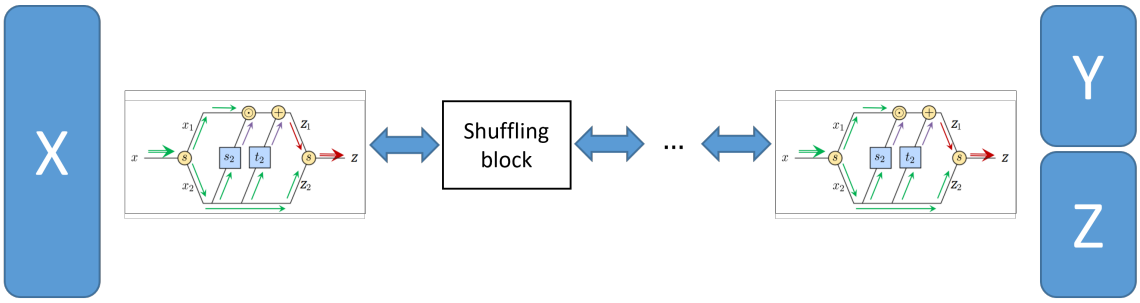


Figure 7.4: INN architecture

where D_x is the dimension of x, D_y is the dimension of y, D_z is the dimension of z. We pad appropriately extra zeros D_{0_x} and $D_{0_{yz}}$. To ensure that the dimensions of the input and output are same, we pad the auxiliary variable z along with the appropriate zero-padding. The augmented input and output vectors become $X_{aug} = [x; 0_x]$ and $Y_{aug} = [y; z; 0_{yz}]$ respectively.

7.2.2 Discriminators

We introduce two discriminators: one for the design space D_X and one for the output space D_Y. Both the discriminators are trained to distinguish the samples provided from the INN-generator and the training data. The discriminators consists of fully-connected networks to output a softmax version of the resultant signal:

$$Disc(X) = softmax(X) \tag{7.13}$$

When converged, the discriminators D_X and D_Y should not be able to tell the difference between data samples coming from the INN-generator and the actual training data.

7.2.3 Training AINN

Algorithm algorithm 4 shows the training process for AINN. The input to the algorithm is the training data and learning rate schedule. The output is a trained model capable of producing directly the inverse surrogate function space. First, we split the data into batches

for training. Inside each training loop, we consider a batch of inputs and output tuple (x, y). The inputs is passed through the INN, generating the output y_{pred} and the latent variable z_{pred}. We find the fitting loss L_y between the training set data y and the predicted output y_{pred}:

$$L_y = MSE(y, y_{pred}) = \frac{1}{B} * (\Sigma_{i=1}^{B}(y_i - y_{i,pred})^2) \tag{7.14}$$

where B is the number of samples in a single batch. Next, we pass the output tuple $[y_{pred}, z_{pred}]$ as an input to the discriminator D_Y. We then find the binary cross entropy (BCE) loss versus true labels:

$$BCE(p, q) = -\frac{1}{B}\Sigma_{i=1}^{B} q * log(p_i) \tag{7.15}$$

where p_i is the probability output of the discriminator and q denotes if the loss is against fake labels or true labels.

Next, we choose the y from the training batch and sample $z \sim p(z)$. This tuple is then passed through the INN in the reverse direction to obtain x_{pred}. Again, we find the fitting MSE loss between the actual and predicted input tuple. We also train the discriminator D_X in a similar fashion. After finding all four losses, they are combined in a linear combination to obtain the total loss L_{total}. In practice, it has been found that the weights $w_y, w_{gen}, w_{D_Y}, w_{fit}, w_{D_X}$ are chosen such that the contribution from each loss component is approximately equal. Having obtained the training loss, we then update the model parameters using gradient descent.

7.3 Numerical Example 1: Patch Antenna

We choose a single patch antenna as a first application to illustrate our model as a proof of concept. A microstrip patch antenna is widely used in portable wireless devices. The output specifications of the antenna are mainly the gain and the center frequency. These metrics depend on the dimensions of the patch and the feed line.

7.3.1 Model Setup

For the inverse design model setup, our design space comprises of the physical dimensions of the patch as well as the feedline. The design space is illustrated in Table 7.1 and is shown in the corresponding Figure 7.5. Here, the inputs to the AINN is $X = \{W_p, L_p\}$ corresponding to various gains G, bandwidth BW and center frequency f_c, forming the output tuple $Y = \{G, f_c\}$. We gather the dataset of 1000 different design points with a 3D EM software. Furthermore, we have $D_x = 2$, $D_y = 2$, $D_z = 6$ and choose $D_{total} = 12$ for this example.

7.3.2 Results

We set aside 800 design tuples for training randomly selected from the data and the remaining 200 designs for evaluation of the model. For this case, we set learning rate $\alpha = 0.01$ and choose an INN with 8 hidden stacked invertible blocks. The discriminator D_X and D_Y each consist of 4 fully-connected layers having [D_{total}, 6, 4, 1] neurons per layer. The loss from the discriminator is used to train the discriminators. The model is run for 100 epochs achieving desirable training loss. To validate our model, we choose a desired performance metric of $Y_{target} = \{G = 6dB, f_c = 140GHz\}$. After the AINN is trained, we can generate the conditional posterior distribution $p(x|y_{target})$. Figure 7.6 shows the joint inverse posterior distribution. As is clear from the 2D contour, the distribution is bi-modal and hence two candidate points are chosen for which the model is most confident about its predictions marked as red stars in the figure. For design verification, we simulate the picked candidate points in 3D EM software and analyze the gain, bandwidth and center frequency of resultant antennas. The performances of the candidate points are shown in Table 7.2. We see that the AINN is able to capture the ambiguity associated with the inverse problem. Both the candidate points are near to the performance spec.

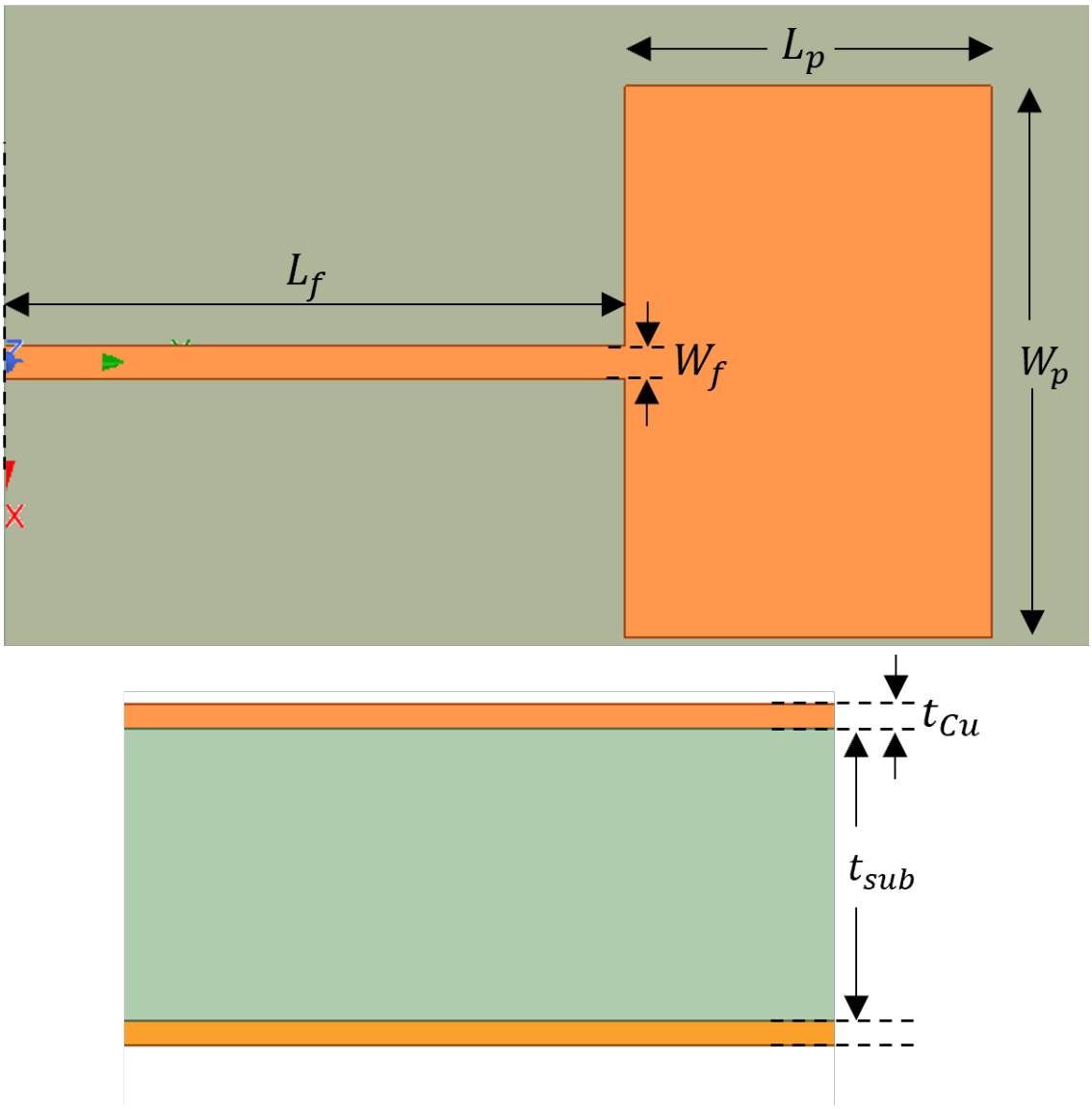


Figure 7.5: Microstrip Patch Antenna Structure

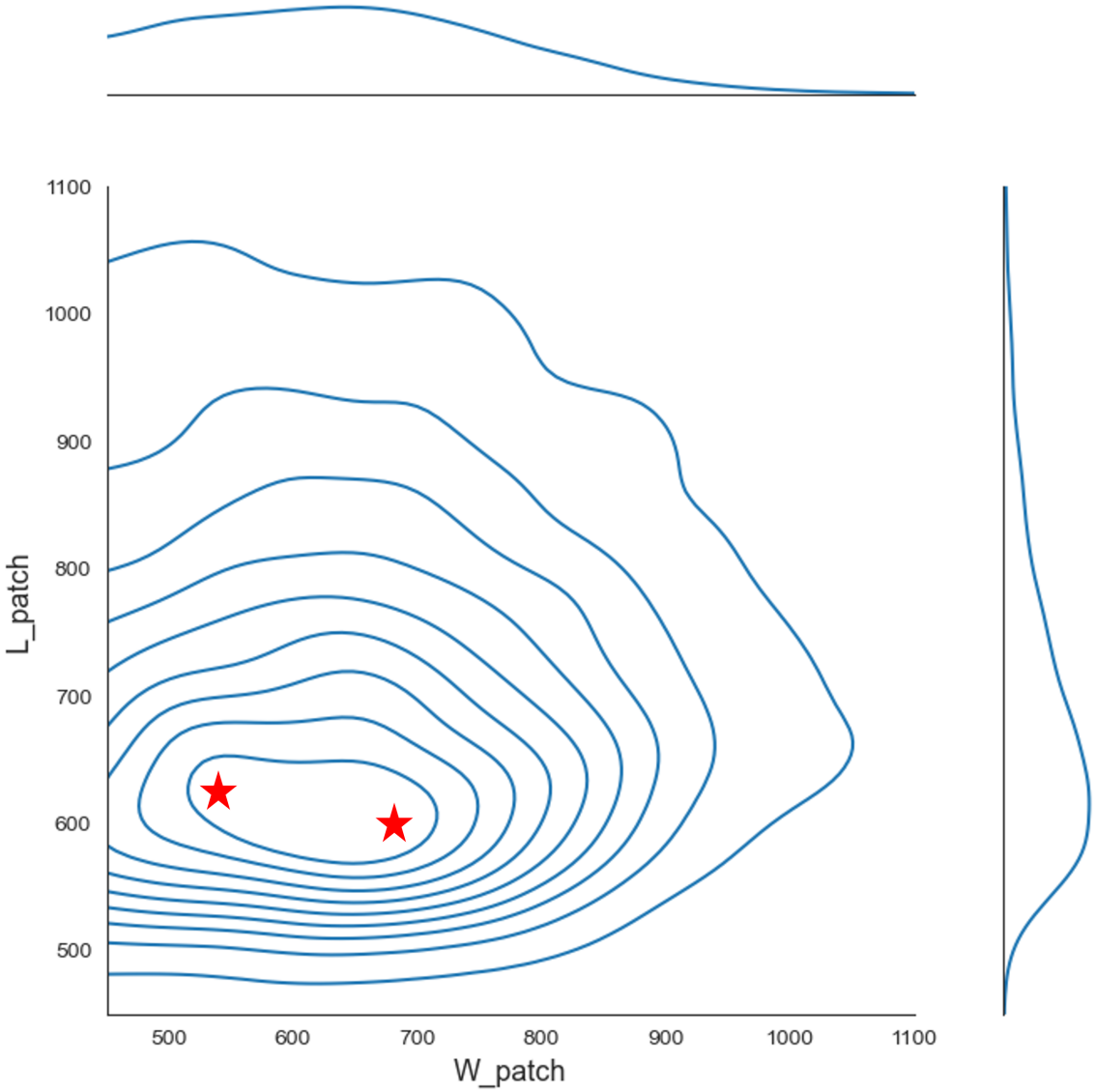


Figure 7.6: Predicted conditional posterior distribution of the design parameters. Candidate points are marked as red stars. Here, $Y_{target} = \{G = 6dB, f_c = 140GHz\}$

7.4 Numerical Example 2: Substrate Integrated Waveguide

Another application chosen for the illustration for inverse design is SIW Interconnect. SIW technology is a promising candidate for interconnects for mmWave applications [72]. At higher frequencies, the effect of parasitics from copper and dielectrics is profound - hence the need to design low-loss interconnect. We illustrate here how AINN can be used for determining the best physical parameters for a target response in D-band (110 GHz - 170 GHz). The SIW considered for this example is shown in Figure 7.7

7.4.1 Model Setup

The structure of the SIW is parameterized by a 10-dimensional space, with their ranges shown in Table 7.3. Hence for the AINN, the design space $X \in R^{10}$. The output Y of the AINN is the full frequency response. We have 600 frequency points uniformly spaced from 110 GHz - 170 GHz. At each frequency point, we have complex-valued 2-port S-parameter matrix. Hence the output shape is (600, 2, 2, 2). Since the S-parameter matrix is symmetric, we consider only S_{11} and S_{21} at each frequency point. We flatten the full frequency response and hence the output dimensionality D_y comes out to be 2400 taking into account the real and imaginary parts. An illustration for the AINN-SIW model setup is shown in Figure 7.8. We have 185 examples for our training set and 25 examples for our evaluation set. We use Ansys HFSS to extract the S-parameters of the SIW to model the transmission losses in this paper.

7.4.2 Results

We run the AINN for 200 epochs, stopping early and keeping the model from overfitting. An adaptive learning rate scheduler is applied to find the minimum loss function. Since, it is easy to overfit the discriminator, we update the discriminator one time for every ten updates of the INN-generator model. Once the model is trained, we can find inverse posterior

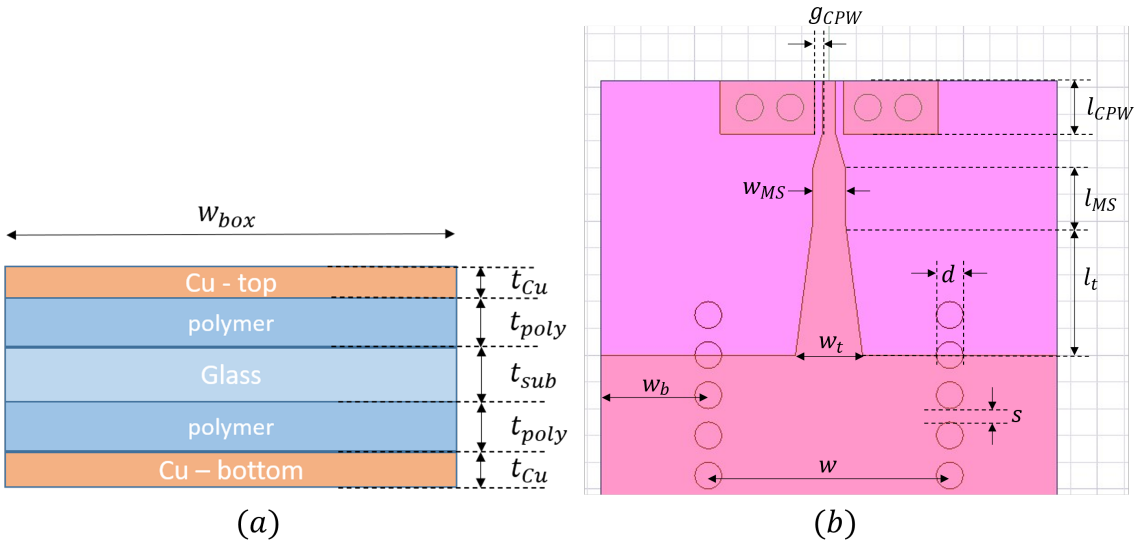


Figure 7.7: SIW structure: (a) stackup (b) half top-view

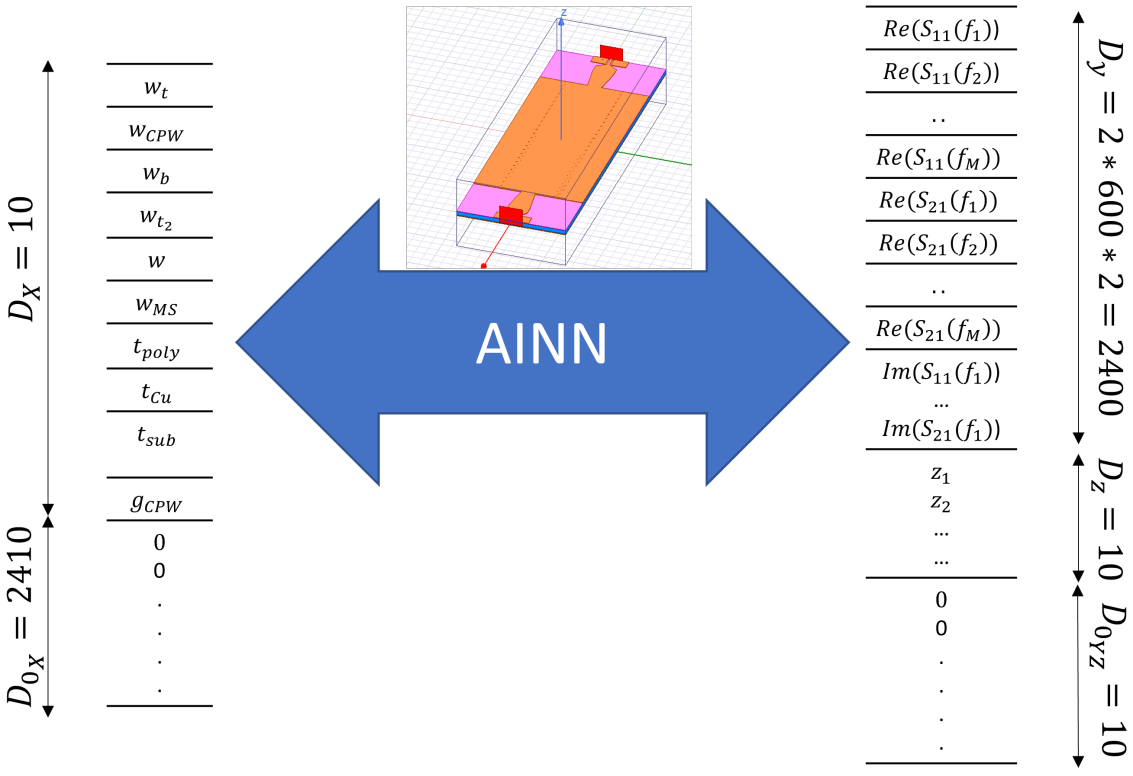


Figure 7.8: AINN-SIW model setup

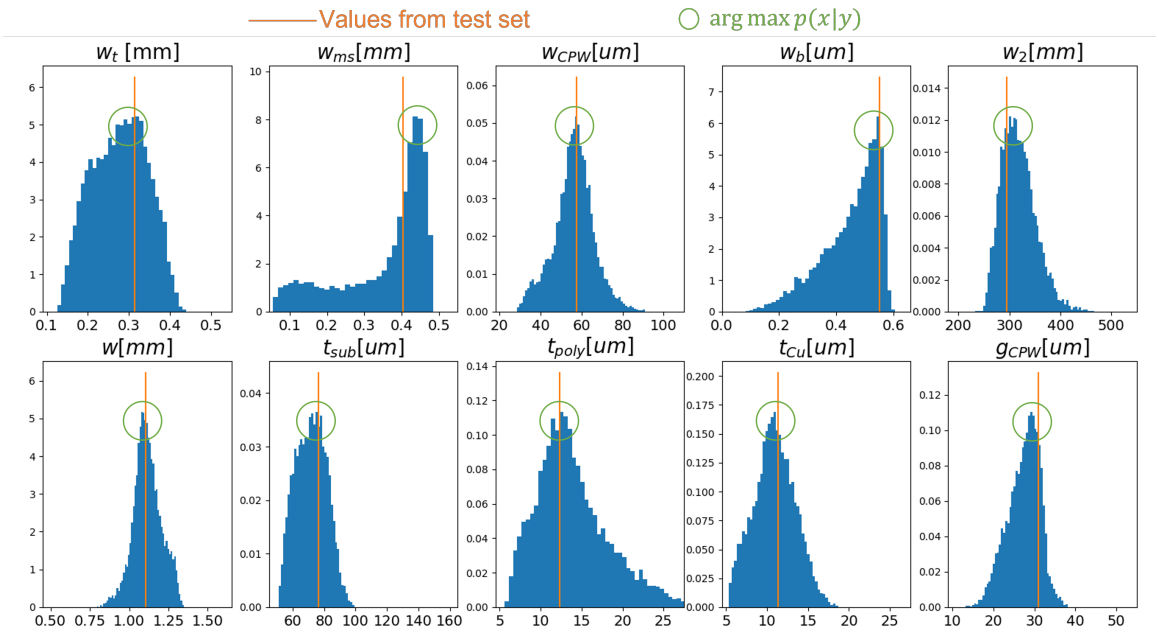


Figure 7.9: Inverse Posterior distributions for SIW design space for Y_{target} shown in Fig.Figure 7.10

distributions for a specified target.

To obtain the joint inverse posterior $p(x|Y_{target})$, we sample the 10-dimensional z-vector 10,000 times. The distributions are shown with normalized densities in Fig.Figure 7.9. The information given to the designer for this task is rich since, we can see how confident the model is in predicting that the combination of the design tuple will produce the desired response. The vertical orange line indicates the values from the test set that produced the given response Y_{target}. Note that the values might not be the only correct values since the transformation from the design space to the response space is not bijective. For verification purposes, we find the most confident design tuple: $X_{pred} = \arg\max_X p(X|Y_{target})$.

To validate the findings of the model, we simulate the predicted design tuple and get $Y_{pred} = T(X_{pred})$ where T is the forward simulation model. In the **??**, we compare the RMSE for the $S - parameter$ matrix averaged over all frequency points for all test cases. We also show the correlation for Y_{pred} and Y_{target} in Figure 7.10. We can see that there is good agreement between the predicted response and the target response.

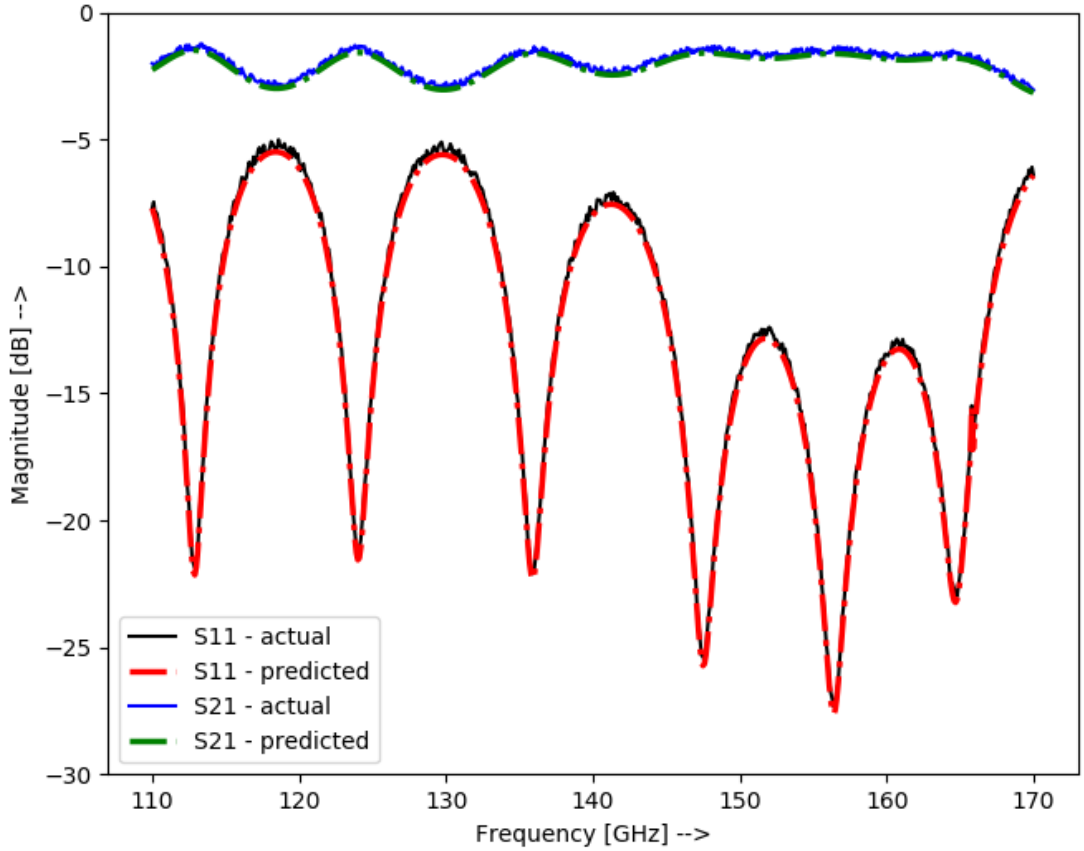


Figure 7.10: Comparison of responses from target response and response simulated from AINN

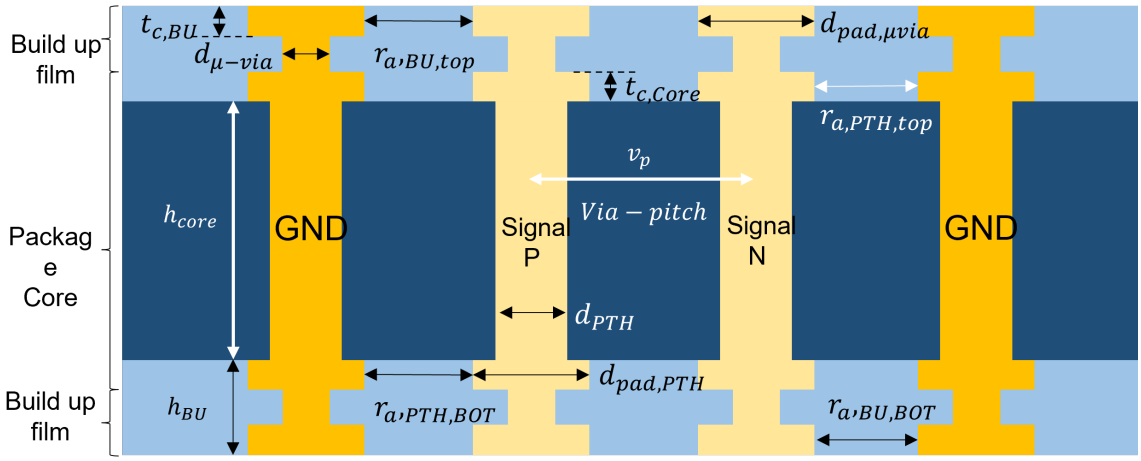


Figure 7.11: Parameters of the differential PTH in package core [63].

7.5 Numerical Example 3: Differential via pair for high-speed signalling

The third application to illustrate the capabilities of our proposed inverse surrogate model is a differential via pair. Such structures are common in high-speed channels. During package design, vias enable a vertical connection between traces and are often crucial to signal transmission. The design space for such a pair must be parameterized to achieve the desired channel bandwidth. The objective for this application is to learn an inverse mapping between the geometrical parameters of the differential pair structure to four-port differential broadband S-parameters. If an efficient inverse mapping is achieved, it can be used to reduce iterative design cycles. We consider the via passing through package thru-hole (PTH) as well as the microvias for connecting the package core to build-up layers. In high-speed design, signal integrity for such models is crucial. The PTH structure is shown in Figure 7.11. After getting the inverse design solutions, we can quantify uncertainty in the S-parameters of the shown structure.

7.5.1 Model Setup

The shown structure is parameterized by a 13-D input design space. Each control parameters of the structure have their minimum and maximum bounds depicted in Table 7.4.

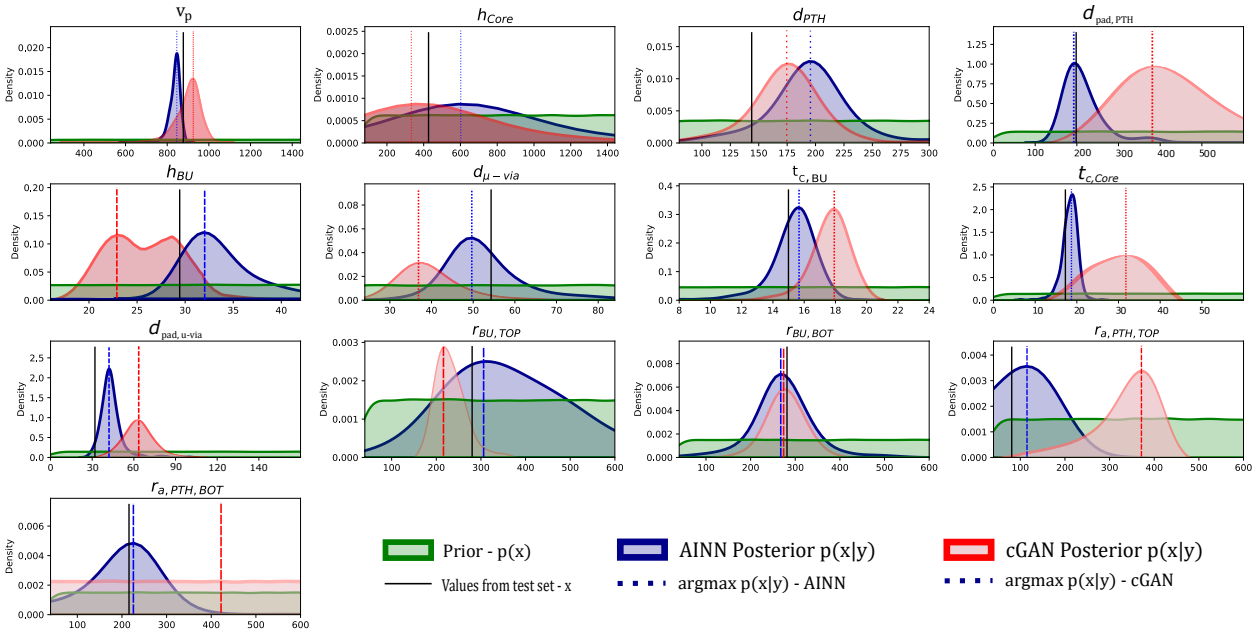


Figure 7.12: Inverse posterior distributions $p(\mathbf{x}|y_{target})$, black vertical line shows values from the test set, y_{target} is as shown in Figure 7.13

Hence, any input design tuple $X \in R^{13}$ has a corresponding response space tuple. The response tuples are four-port scattering (S) parameter matrix from 0.1-100 GHz with steps of 100 MHz. The PTH is a partially reciprocal and symmetric structure, hence, there are only some unique S-parameters and others can be inferred from these ones. We only consider $S_{11}, S_{12}, S_{13}, S_{14}, S_{33}$ and S_{34}, resulting in an output dimension of 6000. Hence, the response space tuple is $Y \in R^{6000}$. We gather our data using Latin Hypercube Sampling (LHS) and obtain total 682 samples. We also obtain the corresponding S-parameters using a commercial 3D EM solver such as Ansys HFSS. For our AINN model, we use 500 of the total data to serve as training data and rest serves for evaluating the accuracy and efficiency of our model. In this case, we use a discriminator D_X as a 4-layer fully-connected neural network consisting of [13, 10, 5, 1] neurons per layer and a response space discriminator D_Y as a 6 layer FCNN consisting of [6000, 500, 255, 130, 25, 1].

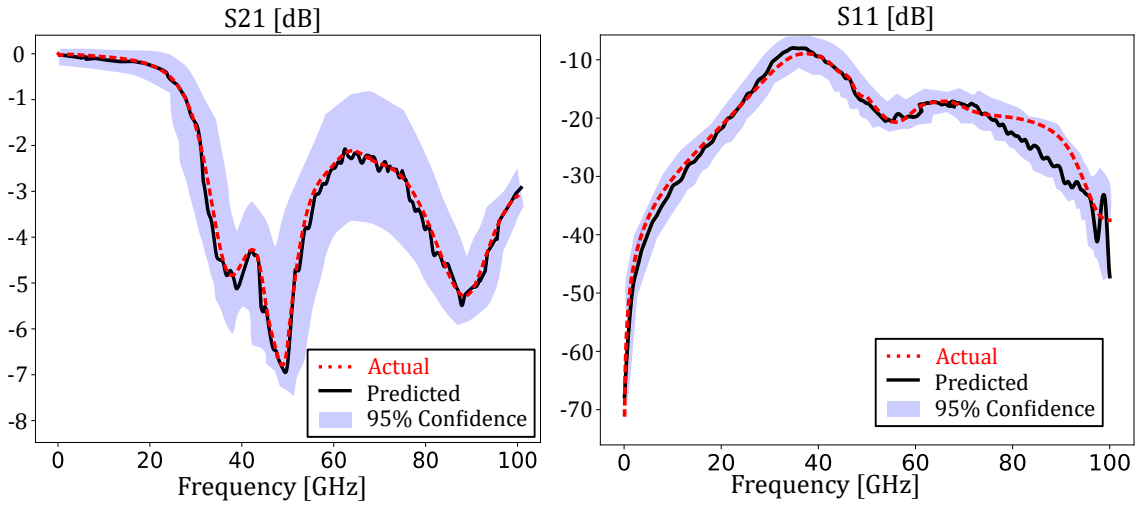


Figure 7.13: Forward simulation results comparison for AINN predictions with 3D EM solvers

7.5.2 Results

We train the whole AINN architecture as shown in algorithm 4 for 200 epochs with an adaptive learning rate scheduler with an initial value of 0.01 using Adam optimizer. After 200 epochs, the model converges until the discriminators cannot distinguish between the real and fake frequency samples. To evaluate our model, we a desired response y_{target} from the test set. We stack the desired response with the latent variable $z \sim p(z)$ for 10,000 times to obtain $x_{pred} = f_{\theta}^{-1}(y, z)$. After doing the reverse pass, we get the joint multidimensional inverse posterior distributions required. Since, we cannot visualize a 13-D probability distribution function, we marginalize individual design space parameters on all dimensions and illustrate those in Figure 7.12. To illustrate how AINN learns these distrubtions, we also show the initial priors. This means before training the AINN thinks that all values in the design space are equally probable. As training happens, the inverse distributions become dense and focus around a certain value. This value is the one that the model is the most confident to produce the desired target. We sample this distribution $p(x|y_{target})$ 10 times and simulate the tuples. The simulated tuples give us a confidence region around the mean prediction. Thus, we can quantify uncertainty in the AINN predictions.

As mentioned, we simulate the chosen input ranges into a forward simulator to obtain confidence intervals. In Figure 7.13, we plot the y_{target} from the test set coming from the 3D EM solver. We compare it with the output from the AINN. We can note that mean of the predicted frequency responses from the INN closely matches the test set values. Specifically, we use the loss normalized mean-squared error metric over each frequency response in the test set:

$$NMSE = \frac{1}{N_d D_y} \times \sum_{d=1}^{D_y} \sum_{n=1}^{N_d} \times \left(\frac{\sum_{m=1}^{N} (S_{n,d}[m] - \hat{S_{n,d}}[m])^2}{\sum_{m=1}^{N} (S_{n,d}[m] - \frac{1}{N} \sum_{m=1}^{N} \hat{S}_{n,d}[m])^2} \right) \quad (7.16)$$

where N_d are the number of evaluation designs for the model and $D_y = 6$ represents the magnitude of the learnt S-parameters. The NMSE value for the proposed approach is 3.3%.

7.5.3 Comparison

Furthermore, we compare other invertible architectures with the proposed AINN in their respective performance criteria in the Table 7.5: (a) feed-forward neural network(FFNN) has a fair amount of network parameters to tune. It is a deterministic model without any guarantees for uniqueness, (b) conditional GAN (cGAN) is a method discussed earlier having a traditional conditional generator and a discriminator that is trained adversarially. (c) invertible neural network [73], that consists of multiple invertible blocks harnessing normalizing flows has less number of parameters and finally the proposed approach. We can also see that AINN has around the number of parameters as the INN including the number of parameters from the disciminator. The training time for this example for different methods is also shown. Moreover, the NMSE for AINN is 9 times lower than the traditional feed-foward neural network.

We also comment on the high-dimensionality of the examples used. This is illustrated in Table 7.6. The compare the inference time for the ML-based AINN with the time taken

by the 3D EM solver to find the optimum tuple.

7.6 Conclusion

High dimensional inverse design poses challenges for modern microwave applications. We present adversarial invertible neural networks that treat the INN as a conditional generator and propose discriminiators at both input and output sides to model high-dimensional probability divergence. We illustrate our approach in three cases: (a) a single patch antenna with 2 dimensions for visualization, (b) a SIW interconnect where design space is directly mapped to S-parameters matrix, and (c) plated-thru hole in package core for modeling high-speed vertical links.

Algorithm 4: Training AINN

Output: Trained model, training loss, inverse mapping

Initialization: Training set TS: (X_{TS}, Y_{TS}), learning_rate: lr, num_epochs, $real\_labels = 1$, $fake\_labels = 0$, $p(z)$

for *i = 1 : num_epochs* **do**

  for $x, y \in (X_{TS}, Y_{TS})$ **do**

    // Forward pass

    $[y_{pred}, z_{pred}] = f_\theta(x)$

    $L_y = MSE(y, y_{pred})$

    // Train forward generative INN

    $d_{pred} = D_Y([y_{pred}; z_{pred}])$

    $L_{Z,gen} = BCE(d_{pred}, true\_labels)$

    // Train Discriminator Y

    $d_1 = D_Y([y; z])$

    $L_{D_{Y,1}} = BCE(d_1, true\_labels)$

    $d_2 = D_Y([y_{pred}; z_{pred}])$

    $L_{D_{Y,2}} = BCE(d_2, fake\_labels)$

    $L_{D_Y} = 0.5 * (L_{D_{Y,1}} + L_{D_{Y,2}})$

    // Reverse pass

    $z \sim p(z)$

    $x_{pred} = f_\theta^{-1}([y; z])$

    $L_{X,fit} = MSE(x, x_{pred})$

    // Train reverse generative INN

    $d_{pred} = D_x(x_{pred})$

    $L_{x,gen} = BCE(d_{pred}, true\_labels)$

    // Train Discriminator X

    $d_1 = D_X(x)$

    $L_{D_{X,1}} = BCE(d_1, true\_labels)$

    $d_2 = D_X(x_{pred})$

    $L_{D_{X,2}} = BCE(d_2, fake\_labels)$

    $L_{D_X} = 0.5 * (L_{D_{X,1}} + L_{D_{X,2}})$

    $L_{total} =$
    $w_y * L_y + w_{gen} * L_{Z,gen} + w_{D_Y} * L_{D_Y} + w_{fit} * L_{X,fit} + w_{gen} * L_{X,gen} + w_{D_X} * L_{D_X}$

    // Backpropagate

    $\forall p \in model.parameters()$

    $p \leftarrow p - lr * \nabla_p L_{total}$

  end

end

Table 7.1: Patch Antenna Design Space Parameters

Parameters		**Unit**	**Min**	**Max**
Patch width	W_p	μm	500	900
Patch length	L_p	μm	500	900
Feed width	W_f	μm	50	
Feed length	L_f	μm	900	
Copper thickness	t_{Cu}	μm	9	
Substrate thickness	t_{sub}	μm	72.5	

Table 7.2: Performance of inverse design candidates for microstrip patch antenna

Performance	**Parameter**			
	W_p (μm)	L_p (μm)	G dB	f_c (GHz)
Design Target	-	-	6	140
Candidate 1	628	534	5.94	140.6
Candidate 2	593	681	6.11	138.2

Table 7.3: SIW Design Space Parameters

Parameters		**Unit**	**Min**	**Max**
Transition width	w_t	mm	0.1	0.5
Microstrip width	w_{MS}	mm	0.05	0.5
CPW width	w_{CPW}	μm	20	100
Out width	w_b	μm	100	600
Transition width II	w_{t_2}	μm	200	500
SIW width	w	mm	0.5	1.5
Polymer thickness	t_{poly}	μm	5	30
Copper thickness	t_{Cu}	μm	5	20
Glass thickness	t_{sub}	μm	50	150
CPW g-plane	g_{CPW}	μm	10	50

Table 7.4: Control Parameters of the PTH Structure

Parameter		**Unit**	**Min**	**Max**
μ-via Diameter	$d_{\mu-via}$	μm	30	70
μ-via Pad Diameter	$d_{pad,\mu-via}$	μm	31	140
BU Layer Thickness	h_{BU}	μm	20	35
μ-via Top Antipad Radius	$r_{BU,TOP}$	μm	100	500
μ-via Bot. Antipad Radius	$r_{BU,BOT}$	μm	100	500
PTH Pitch	v_P	μm	300	1200
Core Thickness	h_{Core}	μm	100	1200
BU Cu Thickness	$t_{c,BU}$	μm	10	20
Core Cu Thickness	$t_{c,Core}$	μm	11	40
PTH Diameter	d_{PTH}	μm	100	250
PTH Pad Diameter	$d_{pad,PTH}$	μm	110	500
PTH Top Antipad Radius	$r_{a,PTH,TOP}$	μm	50	500
PTH Bot. Antipad Radius	$r_{a,PTH,BOT}$	μm	50	500

Table 7.5: Model Comparison for PTH example

Metrics	**FCNN**	**cGAN** [53]	**INN** [73]	**AINN** [This Work]
$NMSE$	9.3%	8%	3.32%	1.11%
Training time	120 min	90 min	78 min	80 min
Inference time	35 ms	90 ms	50 ms	42 ms
Model Size(parameters)	≈152k	≈ 547k	≈ 22k	≈35k
Uniqueness information	No	Yes	Yes	Yes

Table 7.6: Simulation time comparison for AINN with full-wave simulation and Machine Learning(ML) methods

Application ——— **Metrics**		**Patch Antenna**	**Substrate Integrated Waveguide**	**Differential Via pair**
Dimensionality	D_x	2	10	13
	D_y	2	2400	6000
	D_z	4	10	1200
	D_{total}	10	2420	7500
Sample points	**Training**	800	566	500
	Inference	200	120	182
AINN	**Inference time**	23 ms	32 ms	42 ms
EMsolver	**non-ML based**	1143 min	2067 min	3300 min
Test Error		0.08 dB	1.1 dB (MSE)	1.11% (NMSE)

CHAPTER 8
SUMMARY AND FUTURE WORK

8.1 Dissertation Summary

This thesis consists of two main parts. (1) Forward Mapping from the design space to the response space. and (2) Inverse Modeling and Design where we map the response surface to the design space while keeping the problem well-posed.

In **Chapter 3**, a method for causal extrapolation of complex-valued frequency response for distributed structures is presented. The technique uses hidden dependencies in the response to extrapolate it beyond the in-band range. We use recurrent neural networks to learn these dependencies. The hilbert transform layer creates a relation from the real part to the imaginary part, ensuring that the extrapolation is causal. The technique is applied to four scenarios (1) microstrip line circuit (2) Co-planar waveguide measured results (3) Interdigital bandpass filter and (4) Power delivery network, with the last being the most challenging to extrapolate. The PDN provides an accuracy of 0.008 ohms. To address model uncertainty, we employ Bayesian RNN that provides confidence. A quantitative study of cutoff versus confidence bound is performed The model becomes less confident with less training data and in predictions where the frequency points is far off from the cutoff frequency. Furthermore, for a given error, it is possible to extrapolate upto a certain range given by the variance of the predictions.

Chapter 4 presents a machine learning based approach to derive frequency response of a power delivery network and distributed electromagnetic structures in general as a function of their geometrical and material properties. The model architecture consists of a fully connected upsampler which is a feed-forward neural network to produce the code in the high dimensional latent space. The code is fed to a transposed convolutional network to

learn the frequency response at discrete frequency points. Results show that such an architecture performs better than only the fully-connected network approach and saves on computational time and resources in comparison to EM solvers. The proposed approach, while taking more time to train reduces the normalized mean squared error by more than 90% for power delivery applications.

In **Chapter 5**, we harness flow-based invertible neural networks to model inverse mapping of the geometrical and material properties to the impedance envelope spec of a typical power delivery network. The proposed approach shows rich inverse distributions learnt by the model.

Chapter 6, we investigate the inverse design solutions for a high speed channel for 3 architectures: fully-connected neural networks, conditional generative adversarial networks and invertible neural networks. We perform a deterministic as well as stochastic comparison. The quality of the posteriors generated by cGAN and INN are similar to actual posteriors. We conclude that inverse design, in general, can aid the designers to find a suitable input combination. Furthermore, we propose a method to perform uncertainty quantification of frequency response as a function of design space parameters using invertible neural networks for signal integrity applications. Specifically, we illustrate a differential plated-through-hole pair in package core as an example. We provide lower and upper confidence bounds for output 4-port S-parameters. We achieve a normalized mean-squared error of 3.3% on the test set.

In **Chapter 7**, we discuss high dimensional inverse design poses challenges for modern microwave applications. We present adversarial invertible neural networks that treat the INN as a conditional generator and propose discriminiators at both input and output sides to model high-dimensional probability divergence. We illustrate our approach in three cases: (a) a single patch antenna with 2 dimensions for visualization, (b) a SIW interconnect where design space is directly mapped to S-parameters matrix, and (c) plated-thru hole in package core for modeling high-speed vertical links.

8.2 Publications

The material presented in this thesis has resulted in following publications:

Conferences

1. O. W. Bhatti and M. Swaminathan, "Impedance Response Extrapolation of Power Delivery Networks using Recurrent Neural Networks," 2019 IEEE 28th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), Montreal, QC, Canada, 2019, pp. 1-3, doi: 10.1109/EPEPS47316.2019.193198.

2. O. W. Bhatti and M. Swaminathan, "Design Space Extrapolation for Power Delivery Networks using a Transposed Convolutional Net," 2021 22nd International Symposium on Quality Electronic Design (ISQED), 2021, pp. 7-12, doi: 10.1109/ISQED51717.2021.94243

3. O. W. Bhatti, N. Ambasana and M. Swaminathan, "Machine Learning Based Uncertainty Quantification of Extrapolated Design Space and Frequency Response for RF Structures," 2021 IEEE MTT-S International Microwave Symposium (IMS), 2021, pp. 16-19, doi: 10.1109/IMS19712.2021.9574988.

4. S. Han, O. W. Bhatti and M. Swaminathan, "Reinforcement Learning for the Optimization of Decoupling Capacitors in Power Delivery Networks," 2021 IEEE International Joint EMC/SI/PI and EMC Europe Symposium, 2021, pp. 544-548, doi: 10.1109/EMC/SI/PI/EMCEurope52599.2021.9559342.

5. N. Ambasana, O. W. Bhatti, M. Ahadi, M. Swaminathan, X. Yang, P. Roy, W. Becker, "Invertible Neural Networks for High-Speed Channel Design Parameter Distribution Estimation," 2021 IEEE 30th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), 2021, pp. 1-3, doi: 10.1109/EPEPS51341.2021.9609225.

6. O. W. Bhatti, N. Ambasana and M. Swaminathan, "Inverse Design of Power Delivery Networks using Invertible Neural Networks," 2021 IEEE 30th Conference on Elec-

trical Performance of Electronic Packaging and Systems (EPEPS), 2021, pp. 1-3, doi: 10.1109/EPEPS51341.2021.9609211.

7. O. W. Bhatti et al., ”Comparison of Invertible Architectures for High Speed Channel Design,” 2021 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS), 2021, pp. 1-3, doi: 10.1109/EDAPS53774.2021.9657014.

8. O. W. Bhatti, O. Akinwande, and M. Swaminathan, ”Uncertainty Quantification with Invertible Neural Networks for Signal Integrity Applications,” 2022 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO).

9. O. Akinwande, O. W. Bhatti , X. Li, and M. Swaminathan. 2022. Invertible Neural Networks for Design of Broadband Active Mixers. In Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD ’22). Association for Computing Machinery, New York, NY, USA, 145–151. https://doi.org/10.1145/3551901.3556491

10. O. Akinwande, O. W. Bhatti and M. Swaminathan, ”Inverse Design of Embedded Inductor with Hierarchical Invertible Neural Transport Net (HINT),” 2022 IEEE Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)

11. S. Han, O. W. Bhatti and M. Swaminathan, ”Reinforcement Learning for the Optimization of Power Plane Designs in Power Delivery Networks,” 2022 IEEE Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)

Journals

1. S. Han, O. W. Bhatti and M. Swaminathan, ”Computation of Maximum Voltage Droop in Power Delivery Networks,” in IEEE Access, vol. 8, pp. 197875-197884, 2020, doi: 10.1109/ACCESS.2020.3035046.

2. O. W. Bhatti, H. M. Torun and M. Swaminathan, "HilbertNet: A Probabilistic Machine Learning Framework for Frequency Response Extrapolation of Electromagnetic Structures," in IEEE Transactions on Electromagnetic Compatibility, doi: 10.1109/TEMC.2021

3. M. Swaminathan, O.W.Bhatti, Y. Guo, O. Akinwande, E. Huang, "Bayesian Learning for Optimization, Uncertainty Quantification and Inverse Design", Special Issue of TMTT, Feb'22

4. O. W. Bhatti, and M. Swaminathan, "AINNs: Adversarial Invertible Neural Networks for high dimensional inverse design," in IEEE Transactions on Microwave Theory and Techniques [submitted]

5. S. Han, O. W. Bhatti and M. Swaminathan, "Reinforcement Learning for the Optimization of Decoupling Capacitors in Power Delivery Networks," 2021 IEEE International Joint EMC/SI/PI and EMC Europe Symposium, 2021, pp. 544-548, doi: 10.1109/EMC/SI/PI/EMCEurope52599.2021.9559342.

Magazine Articles

1. O. W. Bhatti, N. Ambasana and M. Swaminathan, "Design Space and Frequency Extrapolation: Using Neural Networks," in IEEE Microwave Magazine, vol. 22, no. 10, pp. 22-36, Oct. 2021, doi: 10.1109/MMM.2021.3095706.

8.3 Future Work

In the context of this thesis, there are many streams of interesting future work that can be adopted. In this work, we largely parameterize the design space and find an optimal mapping to the response space. We present CNNs, RNNs, FCNN, INNs for those tasks. One of the important aspects to note is that these architectures are heavily supervised modeling.

One way to move forward is to use a reinforcement learning approach. In such an approach, the model is not told what to do instead, the training agent learns what to do

by itself only by being provided a reward of some kind. For example, consider the SIW structure shown earlier, the design target for S11 might be -20 dB at a frequency of interest. Instead of parameterizing and fixing the design space variables, we can leave the model by a policy and employ Q-networks or policy iteration schemes and provide a reward whenever the model is able to make a move in the right direction. Another application is power plane splitting. Such a problem is a sequential decision-making problem, whereby we do not know the optimal power splitting scheme but we do know that the optimal power planes will have suitable voltage droop and impedance condition satisfied.

Furthermore, To skip annotation time and compute, one can employ unsupervised or at least semi-supervised machine learning techniques. Instead of collecting large amounts of training data to find a good model fit, we need to focus on techniques by which we can minimize the training set size while achieving a generalization error within reason.

Appendices

In this appendix, we introduce some pwoer delivery techniques with regards to work done with ASUStek. I would like to acknowledge Andries Deroo and Yang from ASUS for their help with understanding the problem and providing with automation frameworks.

APPENDIX A

POWER DELIVERY RAILS AND PLANE SPLITTING

As mentioned earlier, power delivery networks need to ensure a clean power supply from the voltage regulator module (VRM) to the IC chip which acts as a current sink. As an illustration, we show the lumped model for the PDN in Figure A.1 [74]. A good power source needs to meet DC power requirement and reduce power fluctuation caused by AC current switch.

In this work, we formulate the problem with finding the ultimate routing given constraints. The proposed model is shown in Figure A.2.

We start from the first two inputs. As shown in Figure A.3, the VRMs are located in a contrained allowed area. The CPU chip is also shown.

The INN model is formulated as shown in Figure A.4.

We run the PDN-INN setup for 100 epochs and show the accumulated results in Figure A.5.

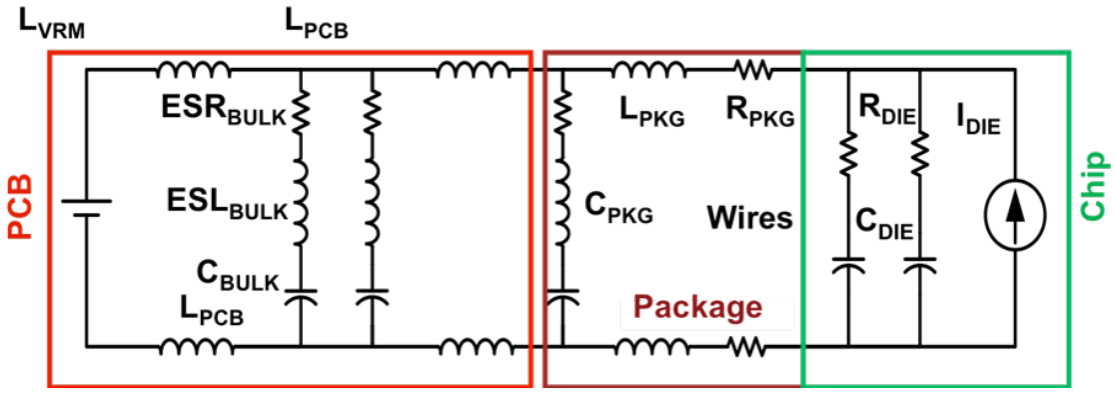


Figure A.1: A typical circuit model for a power delivery network

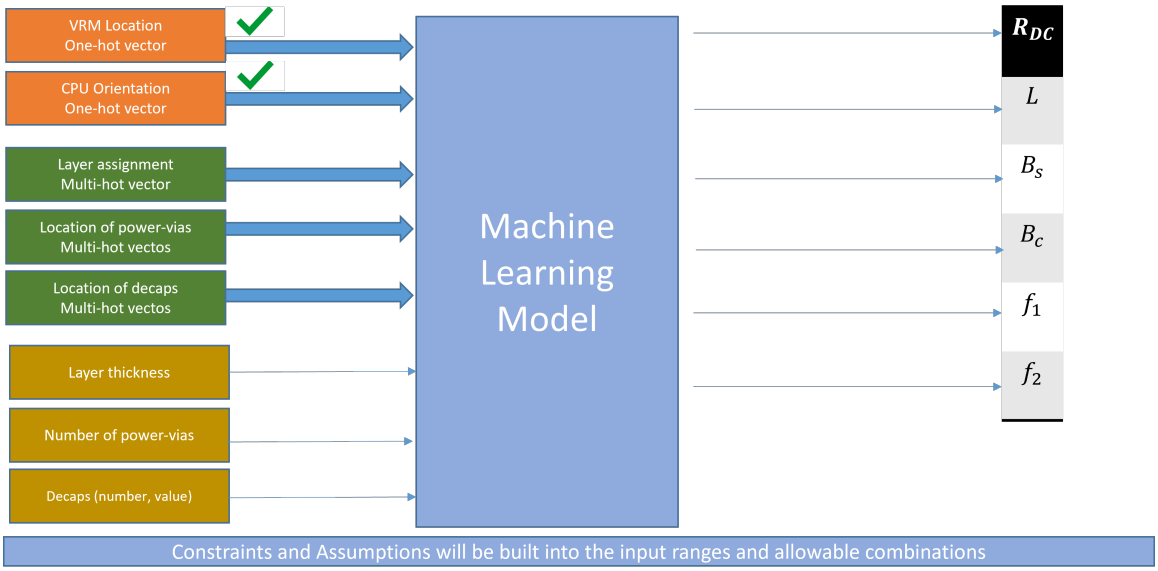


Figure A.2: Proposed input-output problem setup for power delivery application

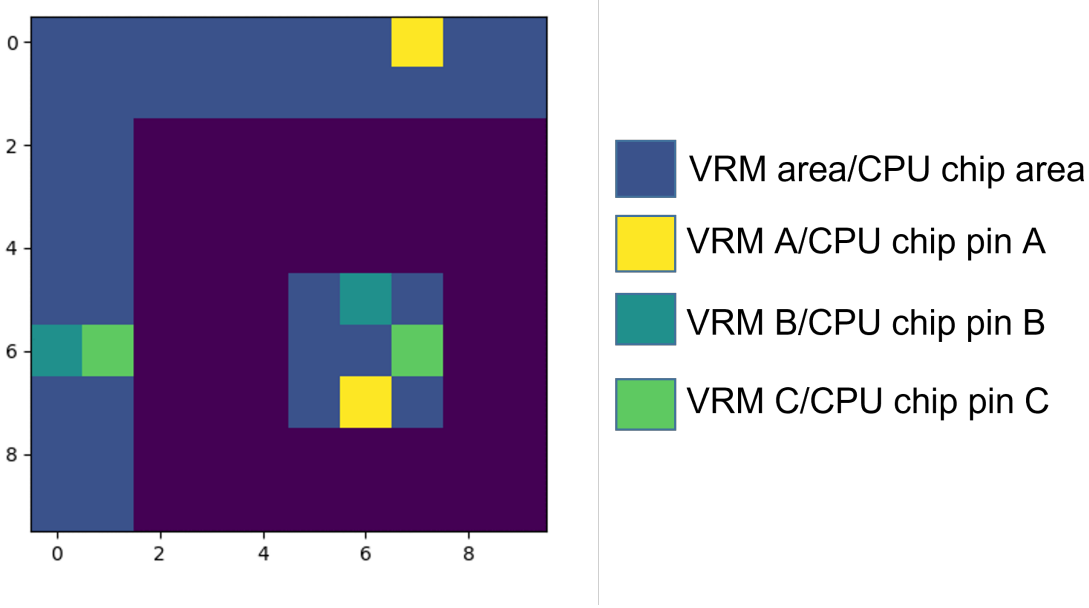


Figure A.3: Proposed input-output problem setup for power delivery application

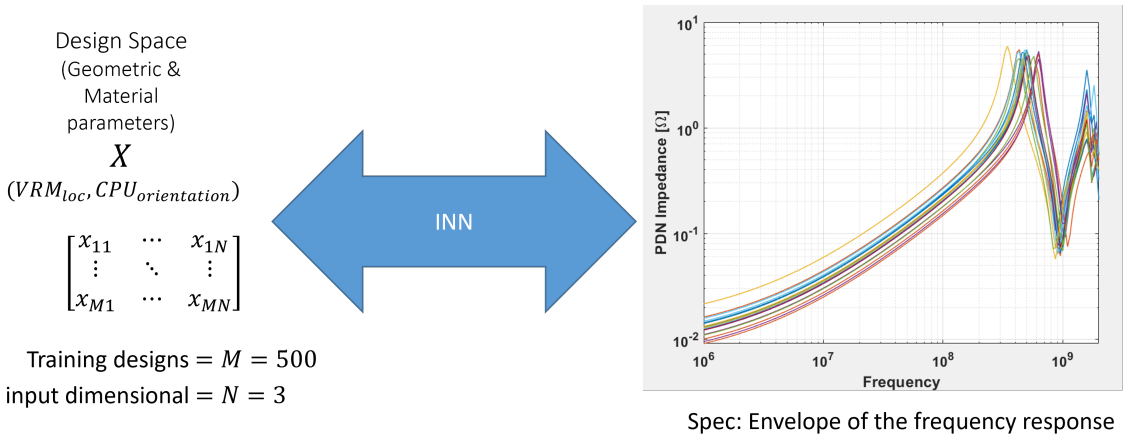
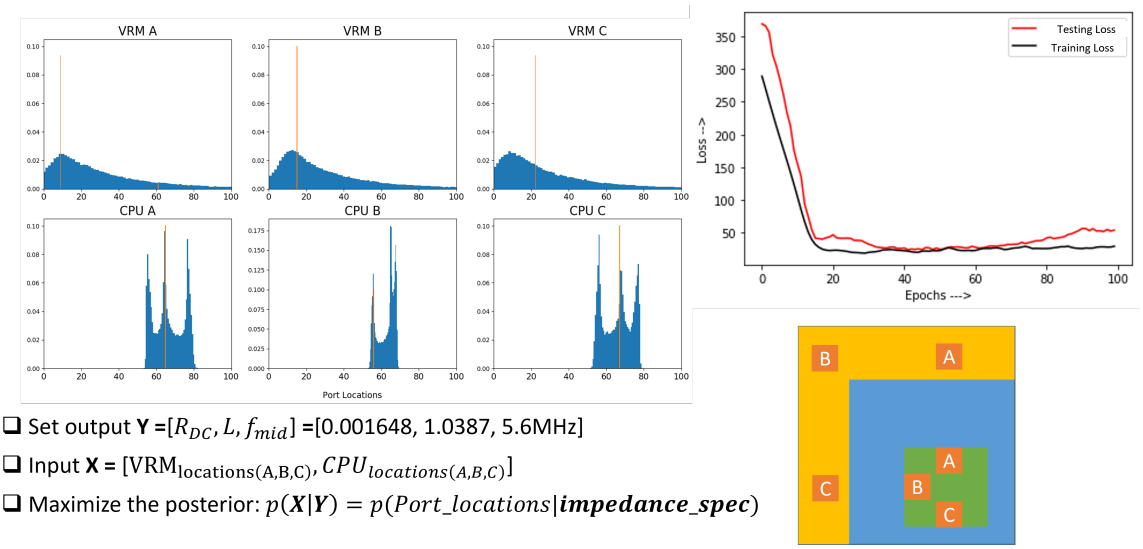


Figure A.4: Proposed input-output problem setup for power delivery application - dimensionality comparison



- ❑ Set output **Y** =$[R_{DC}, L, f_{mid}]$ =[0.001648, 1.0387, 5.6MHz]
- ❑ Input **X** = [$\text{VRM}_{\text{locations(A,B,C)}}$, $CPU_{locations(A,B,C)}$]
- ❑ Maximize the posterior: $p(\boldsymbol{X}|\boldsymbol{Y}) = p(Port\_locations|\boldsymbol{impedance\_spec})$

Figure A.5: Results for PDN problem setup showing probability distributions for VRM areas and CPU chip direction

REFERENCES

[1] B. Mutnury, M. Swaminathan, and J. Libous, "Macromodeling of nonlinear digital i/o drivers," *IEEE Transactions on Advanced Packaging*, vol. 29, no. 1, pp. 102–113, 2006.

[2] Q.-J. Zhang, K. Gupta, and V. Devabhaktuni, "Artificial neural networks for rf and microwave design - from theory to practice," *IEEE Transactions on Microwave Theory and Techniques*, vol. 51, no. 4, pp. 1339–1350, 2003.

[3] W. T. Beyene, "Application of artificial neural networks to statistical analysis and nonlinear modeling of high-speed interconnect systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 1, pp. 166–176, 2007.

[4] Q.-J. Zhang and L. Zhang, "Neural network techniques for high-speed electronic component modeling," in *2009 IEEE MTT-S International Microwave Workshop Series on Signal Integrity and High-Speed Interconnects*, 2009, pp. 69–72.

[5] T. Lu, J. Sun, K. Wu, and Z. Yang, "High-speed channel modeling with machine learning methods for signal integrity analysis," *IEEE Transactions on Electromagnetic Compatibility*, vol. 60, no. 6, pp. 1957–1964, 2018.

[6] C. H. Goay, A. Abd Aziz, N. S. Ahmad, and P. Goh, "Eye diagram contour modeling using multilayer perceptron neural networks with adaptive sampling and feature selection," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 9, no. 12, pp. 2427–2441, 2019.

[7] Y. Liu, T. Lu, J. Y. Kim, K. Wu, and J.-M. Jin, "Fast and accurate current prediction in packages using neural networks," in *2019 IEEE International Symposium on Electromagnetic Compatibility, Signal Power Integrity (EMC+SIPI)*, 2019, pp. 621–624.

[8] H. Ma, E.-P. Li, J. Schutt-Aine, and A. C. Cangellaris, "Deep learning method for prediction of planar radiating sources from near-field intensity data," in *2019 IEEE International Symposium on Electromagnetic Compatibility, Signal Power Integrity (EMC+SIPI)*, 2019, pp. 610–615.

[9] M. Swaminathan, H. M. Torun, H. Yu, J. A. Hejase, and W. D. Becker, "Demystifying machine learning for signal and power integrity problems in packaging," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 10, no. 8, pp. 1276–1295, 2020.

[10] K. K. Samanta, "3d/multilayer heterogeneous integration and packaging for next generation applications in millimeter-wave and beyond," in *2017 IEEE MTT-S International Microwave and RF Conference (IMaRC)*, 2017, pp. 294–297.

[11] D. S. Green, C. L. Dohrman, J. Demmin, Y. Zheng, and T.-H. Chang, "A revolution on the horizon from darpa: Heterogeneous integration for revolutionary microwavemillimeter-wave circuits at darpa: Progress and future directions," *IEEE Microwave Magazine*, vol. 18, no. 2, pp. 44–59, 2017.

[12] D. Montgomery, *Design and analysis of experiments*. Hoboken, NJ: Wiley, 2020.

[13] A. Norman, D. Shykind, M. Falconer, and K. Ruffer, "Application of design of experiments (doe) methods to high-speed interconnect validation," *Electrical Performance of Electrical Packaging (IEEE Cat*, no. 03TH8710), pp. 15–18, 2003.

[14] V. Sathanur, V. Jandhyala, and H. Braunisch, "A hierarchical simulation flow for return-loss optimization of microprocessor package vertical interconnects," *IEEE Transactions on Advanced Packaging*, vol. 33, no. 4, pp. 1021–1033, Nov. 2010.

[15] E. Matoglu, M. Swaminathan, M. Cases, N. Pham, and D. Araujo, "Design space exploration of high-speed busses using statistical methods," *Electrical Performance of Electrical Packaging (IEEE Cat*, no. 03TH8710), pp. 19–22, 2003.

[16] E. Matoglu, N. Pham, D. Araujo, M. Cases, and M. Swaminathan, "Statistical signal integrity analysis and diagnosis methodology for high-speed systems," *IEEE Transactions on Advanced Packaging*, vol. 27, no. 4, pp. 611–629, Nov. 2004.

[17] N. Singh, "Swarm intelligence for electrical design space exploration," in *2007 IEEE Electrical Performance of Electronic Packaging*, Atlanta, GA, 2007, pp. 21–24.

[18] C. Wesley, B. Mutnury, N. Pham, E. Matoglu, and M. Cases, "Electrical design space exploration for high speed servers," in *Proceedings 57th Electronic Components and Technology Conference*, Reno, NV, 2007, pp. 1748–1753.

[19] P. J., S. D., and B. G, "Optimization strategies in design space exploration," in *Handbook of Hardware/Software Codesign*, H. S. and T. J, Eds., Dordrecht: Springer, 2017.

[20] M. Arumugam, M. Rao, and A. W. Tan, "A novel and effective particle swarm optimization like algorithm with extrapolation technique," *Applied Soft Computing*, vol. 9, no. ue 1, pp. 308–320, 2009.

[21] R. Adve, T. Sarkar, S. Rao, E. Miller, and D. Pflug, "Application of the cauchy method for extrapolating/interpolating narrowband system responses," *IEEE Trans-*

actions on Microwave Theory and Techniques, vol. 45, no. 5, pp. 837–845, May 1997.

[22] S. Narayana, "Interpolation/extrapolation of frequency domain responses using the hilbert transform," *IEEE Transactions on Microwave Theory and Techniques*, vol. 44, no. 10, pp. 1621–1627, Oct. 1996.

[23] S. Ho and M. Xie, "The use of arima models for reliability forecasting and analysis," *Computers Industrial Engineering*, vol. 35, no. 1, pp. 213–216, 1998.

[24] W. Na, W. Liu, L. Zhu, F. Feng, J. Ma, and Q. Zhang, "Advanced extrapolation technique for neural-based microwave modeling and design," *IEEE Transactions on Microwave Theory and Techniques*, vol. 66, no. 10, pp. 4397–4418, Oct. 2018.

[25] R. Kim, J. Doppa, and P. Pande, "Machine learning for design space exploration and optimization of manycore systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD*, San Diego, CA, 2018, pp. 1–6.

[26] S. Sen and N. Imam, "Machine learning based design space exploration for hybrid main-memory design," in *Proceedings of the International Symposium on Memory Systems (MEMSYS '19*, DOI : NY, USA: Association for Computing Machinery, 2019, pp. 480–489.

[27] B. Ozisikyilmaz, G. Memik, and A. Choudhary, "Efficient system design space exploration using machine learning techniques," in *45th ACM/IEEE Design Automation Conference*, Anaheim, CA, 2008, pp. 966–969.

[28] M. Larbi, R. Trinchero, F. Canavero, P. Besnier, and M. Swaminathan, "Analysis of parameter variability in integrated devices by partial least squares regression," in *2020 IEEE 24th Workshop on Signal and Power Integrity (SPI*, Cologne, Germany, 2020, pp. 1–4.

[29] D. Xu, Y. Shi, I. Tsang, Y. Ong, C. Gong, and X. Shen, "Survey on multi-output learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 7, pp. 2409–2429, Jul. 2020.

[30] *Bayesian analysis #1: Concepts*, https://kevintshoemaker.github.io/NRES-746/LECTURE6.html, (Accessed on 03/16/2022).

[31] M. Fortunato, C. Blundell, and O. Vinyals, *Bayesian Recurrent Neural Networks"*. NeurIPS, 2017.

[32] H. M. Torun, "Machine learning based design and optimization for high-performance semiconductor packaging and systems," Ph.D. dissertation, Georgia Institute of Technology, 2020.

[33] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[34] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, 3. MIT press Cambridge, MA, 2006, vol. 2.

[35] D. Duvenaud, "Automatic model construction with gaussian processes," Ph.D. dissertation, University of Cambridge, 2014.

[36] R. M. Neal, "Slice sampling," *The annals of statistics*, vol. 31, no. 3, pp. 705–767, 2003.

[37] H. M. Torun, J. A. Hejase, J. Tang, W. D. Beckert, and M. Swaminathan, "Bayesian active learning for uncertainty quantification of high speed channel signaling," in *2018 IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, IEEE, 2018, pp. 311–313.

[38] *Normalizing flows*, https://deepgenerativemodels.github.io/notes/flow/#:~:text=The%20name%20%E2%80%9Cnormalizing%20flow%E2%80%9D%20can,create%20more%20complex%20invertible%20transformations., Accessed: 2022-09-14.

[39] S. Hochreiter and J. Schmidhuber, *Long short-term memory" joural of neural computation*, 1997.

[40] O. Bhatti and M. Swaminathan, "Impedance response extrapolation of power delivery networks using recurrent neural networks," in *2019 IEEE 28th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS*, Montreal, QC, Canada, 2019, pp. 1–3.

[41] D. Grujić, "Numerical hilbert transform algorithm for causal interpolation of piecewise polynomial even and odd functions," *IEEE Transactions on Microwave Theory and Techniques*, vol. 65, no. 6, pp. 2000–2007, Jun. 2017.

[42] H. Torun, A. Durgun, K. Aygün, and M. Swaminathan, "Causal and passive parameterization of s-parameters using neural networks," *IEEE Transactions on Microwave Theory and Techniques*, vol. 68, no. 10, pp. 4290–4304, Oct. 2020.

[43] M. Fortunato, C. Blundell, and O. Vinyals, *Bayesian Recurrent Neural Networks"*. NeurIPS, 2017.

[44] O. Bhatti, H. Torun, and M. Swaminathan, "Machine learning framework for extrapolation of frequency response"," *IEEE Transactions on Electromagnetic Compatability*, 2020 (under review).

[45] J.-H. Kim and M. Swaminathan, "Modeling of irregular shaped power distribution planes using transmission matrix method," *IEEE Transactions on Advanced Packaging*, vol. 24, no. 3, pp. 334–346, 2001.

[46] J. Kim *et al.*, "Chip-package hierarchical power distribution network modeling and analysis based on a segmentation method," *IEEE Transactions on Advanced Packaging*, vol. 33, no. 3, pp. 647–659, 2010.

[47] L. Y., H. P., B. L., and B. Y, "Object recognition with gradient-based learning," in *Shape, Contour and Grouping in Computer Vision. Lecture Notes in Computer Science*, vol. 1681, Berlin, Heidelberg: Springer, 1999.

[48] O. W. Bhatti and M. Swaminathan, "Design space extrapolation for power delivery networks using a transposed convolutional net," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 7–12.

[49] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, arXiv preprint arXiv:1603.07285. 2016 Mar 23.

[50] M. Ali, "First demonstration of compact, ultra-thin low-pass and bandpass filters for 5g small-cell applications," *IEEE Microwave and Wireless Components Letters*, vol. 28, no. 12, pp. 1110–1112, Dec. 2018.

[51] H. Yu, H. M. Torun, M. U. Rehman, and M. Swaminathan, "Design of siw filters in d-band using invertible neural nets," in *2020 IEEE/MTT-S International Microwave Symposium (IMS)*, 2020, pp. 72–75.

[52] *Latin hypercube sampling*, Dec. 2021.

[53] I. Goodfellow, "Generative adversarial nets"," in *Advances in Neural Information Processing Systems 27*, 2014.

[54] D. Tait and T. Damoulas, *Variational autoencoding of pde inverse problems*.

[55] L. Ardizzone, J. Kruse, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks"," in *International Conference on Learning Representations (ICLR*, 2019.

[56] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," in *International Conference on Learning Representations (ICLR*, 2016.

[57] H. Torun *et al.*, "Design space exploration of power delivery in heterogeneous integration," *GOMACTECH*, Mar. 2019.

[58] *Matlab based tool for pdn impedance analysis*, accessed as of June 7th, 2021.

[59] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989.

[60] S. Ferrari and R. Stengel, "Smooth function approximation using neural networks," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 24–38, 2005.

[61] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. arXiv: 1411.1784.

[62] N. Ambasana *et al.*, "Invertible neural networks for high-speed channel design parameter distribution estimation," in *2021 IEEE 30th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, 2021, pp. 1–3.

[63] H. M. Torun, A. C. Durgun, K. Aygün, and M. Swaminathan, "Enforcing causality and passivity of neural network models of broadband s-parameters," in *2019 IEEE 28th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, 2019, pp. 1–3.

[64] O. W. Bhatti, O. Akindwande, and M. Swaminathan, "Ainns: Adversarial invertible neural networks for high-dimensional inverse design," *IEEE Transactions of Microwave Theory and Techniques*, 2022, submitted.

[65] H. Kabir, Y. Wang, M. Yu, and Q.-J. Zhang, "Neural network inverse modeling and applications to microwave filter design," *IEEE Transactions on Microwave Theory and Techniques*, vol. 56, no. 4, pp. 867–879, 2008.

[66] Y. Tang *et al.*, "Generative deep learning model for inverse design of integrated nanophotonic devices," *Laser & Photonics Reviews*, vol. 14, no. 12, p. 2 000 287, 2020.

[67] S. So, T. Badloe, J. Noh, J. Bravo-Abad, and J. Rho, "Deep learning enabled inverse design in nanophotonics," *Nanophotonics*, vol. 9, no. 5, pp. 1041–1057, 2020.

[68] I. O. Tolstikhin, B. K. Sriperumbudur, and B. Schölkopf, "Minimax estimation of maximum mean discrepancy with radial kernels"," in *Advances in Neural Information Processing Systems 29*, 2016.

[69] O. W. Bhatti *et al.*, "Comparison of invertible architectures for high speed channel design," in *2021 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS)*, 2021, pp. 1–3.

[70] A. Ramdas, S. J. Reddi, B. Póczos, A. Singh, and L. Wasserman, "On the decreasing power of kernel and distance based nonparametric hypothesis tests in high dimensions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

[71] G. Peskir, "A change-of-variable formula with local time on curves," *Journal of Theoretical Probability*, vol. 18, no. 3, pp. 499–535, 2005.

[72] M. Yi *et al.*, "Surface roughness modeling of substrate integrated waveguide in d-band," *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 4, pp. 1209–1216, 2016.

[73] O. W. Bhatti, O. Akinwande, and M. Swaminathan, "Uncertainty quantification with invertible neural networks for signal integrity applications," in *2022 IEEE MTT-S International Conference on Electromagnetic and Multiphysics Modeling and Optimization (NEMO2022)*, accepted.

[74] R. Sjiariel, R. Enjiu, J. Costa, and M. Perotoni, "Power integrity simulation of power delivery network system," in *2015 SBMO/IEEE MTT-S International Microwave and Optoelectronics Conference (IMOC)*, 2015, pp. 1–5.

VITA

Osama Waqar Bhatti (Graduate Student Member, IEEE) received the bachelor's degree in electrical engineering from the National University of Sciences and Technology, Islamabad, Pakistan, in 2017. He is currently pursuing the Ph.D. degree at the Georgia Institute of Technology, Atlanta, GA, USA.,His current research interests include designing machine learning algorithms for signal and power integrity applications.,Mr. Bhatti was a recipient of the Best Paper Award at the 22nd International Symposium on Quality Electronic Design (ISQED'21).