NEURAL NETWORKS BASED SURROGATE MODELS FOR QUANTIFICATION OF GAP AND OVERLAP DEFECTS IN TOW STEERED COMPOSITES

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computational Science

---

by

John Charles Crow

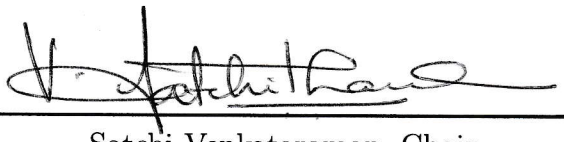
Fall 2022

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

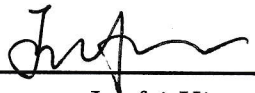
Thesis of John Charles Crow:

NEURAL NETWORKS BASED SURROGATE MODELS FOR QUANTIFICATION OF GAP AND OVERLAP DEFECTS IN TOW STEERED COMPOSITES



Satchi Venkataraman, Chair
Department of Aerospace Engineering and Engineering Mechanics

Christopher P. Paolini
Department of Electrical and Computer Engineering



Junfei Xie
Department of Electrical and Computer Engineering

December 2, 2022.

Approval Date

DEDICATION

Dedicated to my wife.

ABSTRACT OF THE THESIS

NEURAL NETWORKS BASED SURROGATE MODELS FOR QUANTIFICATION OF GAP AND OVERLAP DEFECTS IN TOW STEERED COMPOSITES

by
John Charles Crow
Master of Science in Computational Science
San Diego State University, 2022

Advanced carbon fiber reinforced polymeric composite (CFRP) materials possess high strength and stiffness properties and are lightweight. CFRP materials allow spatial tailoring of material stiffness and strength to meet the load requirements. Such spatially tailored material properties can be constructed by tow steering using Automated Fiber Placement (AFP) manufacturing. Parametrization of the material orientations for designing tow steered composites require a large number of design variables and result in non-convex optimization problems. Fiber or tow paths are generated from the optimized material orientation fields, as a post processing step, using tow placement path planning algorithms.

Curvilinear fiber paths develop overlaps and gaps due to the difference of the curvatures mismatched between fiber tows. These gaps and overlaps decrease the strength of the composite structure. Optimization of composites for tow steered curvilinear paths therefore require manufacturing constraints that limit the overlaps and gaps. This thesis investigated Deep Neural Networks for developing surrogate models of the manufacturing constraints (overlaps and gaps) to replace the using of path planning algorithms in the design optimization directly.

Feed Forward Neural Network (FFNN), Convolution Neural Network (CNN) and Recurrent Neural Networks (RNN) were investigated to determine the most appropriate model for the manufacturing constraint approximation. The input variables are the fiber orientation angles used to develop a spatially varying material orientation and the outputs are the areas of overlap and gap developed by tow steering. The methods are demonstrated for two examples; a flat rectangular plate (16 variables) and a rectangular plate with an elliptical hole (24 input variables).

For the first example the RNN had the best prediction accuracy. For the second example, the accuracy of all DNN models were significantly less accurate compared to the first. However, both the FFNN and RNN had similar accuracy. The decreased prediction accuracy in the latter case is due to the sparsity of the sampling used and the highly non-linear and noisy nature of the output function. The current results from the flat plate problem demonstrate that Neural Networks can provide a computationally efficient and accurate surrogate model to implement manufacturing constraints for tow steered composites.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PAGE

GLOSSARY

AFP Automated Fiber Placement.

CAD Computer Aided Design.

CCD Central Composite Design.

CNN Convolutional Neural Network.

DCEL Double Connected Edge List.

DNN Deep Neural Network.

DOE Design of Experiments.

FFNN Feed Forward Neural Network.

LHS Latin Hyper-Cube Sampling.

LSTM Long Short-term Memory.

MAE Mean Absolute Error.

ML Machine Learning.

MSE Mean Squared Error.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

ACKNOWLEDGMENTS

I would like to acknowledge Chris Minaya whose simulation program is the basis for my surrogate model. I would also like to acknowledge the helpful advice of and the fruitful discussions with Dr. Hammerand. Last but not least thank you to my advisor Professor Satchi Venkataraman.

CHAPTER 1
INTRODUCTION

Neural networks are useful as surrogates in engineering design workflows. They can meaningfully speed up optimization processes with a reasonable loss in accuracy. In the optimization process, they can help provide constraints of factors that can not be defined mathematically such as manufacturing constraints.

Research in design optimization seeks to reduce computational burden, create effective and efficient Design of Experiments, and develop optimization approaches to solve nonlinear, stochastic, or ill-posed problems [33]. This research attempts to utilize Machine Learning techniques to surrogate and reduce the computational burden of a manufacturing simulation program providing manufacturing based constraints to a composite materials optimization. This design process is created for composites manufactured by Automated Fiber Placement machines.

While composite fiber reinforced materials are not a brand new material, their popularity in the past 2 decades has increased as improved manufacturing capabilities have led to increased used in both government and commercial applications. Their high strength to weight ratio make them particularly useful in aerospace applications.

Automated Fiber Placement is a relatively new manufacturing technique where a machine can place strips, referred to as tows, of composite material along curvilinear paths. This allows for greater control of the strength and stress properties of the composite material. Ultimately this results in increased performance with a further decrease in weight.

The drawback to this method is the design and manufacturing process is more complex than traditional approaches. In traditional design a single fiber orientation is used per ply. Multiple plies are layered to achieve a design stiffness and thus strength performance. A variable fiber orientation introduces multiple optimization variables per ply and thus greatly increases the computational cost of the optimization problem. The Automated Fiber Placement machines introduce defects and flaws into the composite structure. An optimal design computed by traditional means may not be manufacturable by the AFP machine. Gaps and overlaps between the tows are possible and are the focus of this work.

In order to provide a feedback constraint to the structural optimization process, a tow path planning simulation algorithm was designed. This simulation helps quantify the manufacturability of the designs created by the structural optimization.

The optimization program passes the fiber orientations on a coarse mesh to the simulation algorithm. These nodal angles are then interpolated on a fine mesh. The overlaps and gaps are calculated on a fine element mesh by using an Eulerian approach to trace trajectories that follow the vector field direction. The tow is made in small incremental steps, marking all elements it traverses on the path as a filled cell. Multiple passes by parallel shifting the initial seed point develops tow trajectories for the entire path. Once an initial set of tow paths are determined overlapped tows are cut and large gap regions are filled in. This incremental path filling approach is computationally expensive. The selection of seeding locations and how to cut overlapped tows introduce non-linearities into the output and are difficult to define mathematically.

Due to the computational expense of the simulation algorithm, we researched the viability of creating a surrogate model for the manufacturing simulation. In the aerospace community, researchers have traditionally used techniques such as Polynomial Response surfaces, Kriging methods, and Radial Basis Functions as surrogates [33].

Due to the nonlinear nature of the simulation algorithm we explored Machine Learning techniques to create surrogate models to predict the gaps and overlaps. Deep neural networks were used to solve the multi-dimensional regression problem. The input to both the Machine Learning Models and Simulation algorithm are the material orientation angles specified on a coarse mesh, denoted as $\Phi = \Phi_{1,1}, \Phi_{1,2}, \ldots, \Phi_{n,n}$ and the output is the total gap and overlap area of the ply as a fraction of the total area $\omega = [A_G, A_O], s.t. A_{G,O} \in [0, 1]$. The simulation detailed above was used to create the sample data sets. Three independent data sets were created for training, validation, and testing of the neural networks. Each data set is a Latin Hypercube Sampling of the input space.

While Machine Learning models are expensive to train [33], they can be trained independently and prior to the overall optimization. The goal of the model would be to return an estimate of ω within some accuracy tolerance while requiring less computational resources and time than the path planning simulation algorithm.

CHAPTER 2
BACKGROUND

In this section we provide background on Advance Composite materials in order to provide context for our simulation algorithm. We address the development of the Automated Fiber Placement process and the attempts to qualify and quantify the manufacturability of different optimization approaches. Then we discuss Design of Experiment techniques used to create the datasets used to train the ML models. In the next chapter we will discuss the Machine Learning concepts of Deep neural networks in order to provide background to the surrogate models.

2.1 Composites

Common structural materials, including metal alloys, ceramics, and polymers are monolithic in nature. The material uniformity in composition gives homogeneity. Isotropy means uniformity of mechanical properties regardless of the direction in which it is measured. Composites materials are heterogeneous and often are not isotropic and are not uniform in their construction [34].

Composites are made by reinforcing a material with another material. Advanced composite fiber materials are composed of a matrix, such as resins or ceramics, and a high-performance fiber reinforcement, such as carbon fibers or aromatic polyamide fibers [34, 40, 47]. The matrix material provides the binding and form while the material that composes the strengthening fibers are called the reinforcement. These 2 main segments can be easily seen in Figure 2.1.

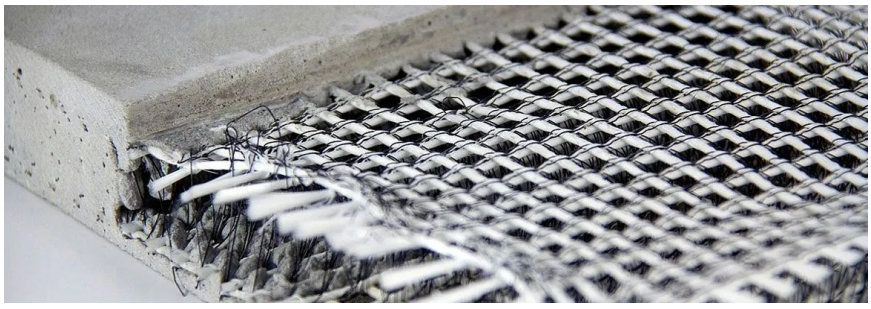


Figure 2.1. A composite material composed of a matrix and reinforcing fibers. [Y. Nawib, Classification and applications of textile composite materials: Part 1, Dec 2009.]

Composites principally serve a dual purpose—increasing the stiffness while decreasing the overall weight of the structure [34]. This strength to weight ratio is highly valuable in engineering applications where weight is an important factor, especially in the fields of aerospace engineering. Advanced Composite materials are representing a larger percentage of aerospace materials. As can be seen in Figure 2.2, composites have now become the material of choice in aircraft manufacturing. The reduction in weight allows for increased efficiencies in fuel/energy consumption. Over the past two decades, commercial fiber placement systems have become widely used to manufacture complex aerospace composite structures, resulting in associated reductions in touch labor time, material wastage,and part counts [46].

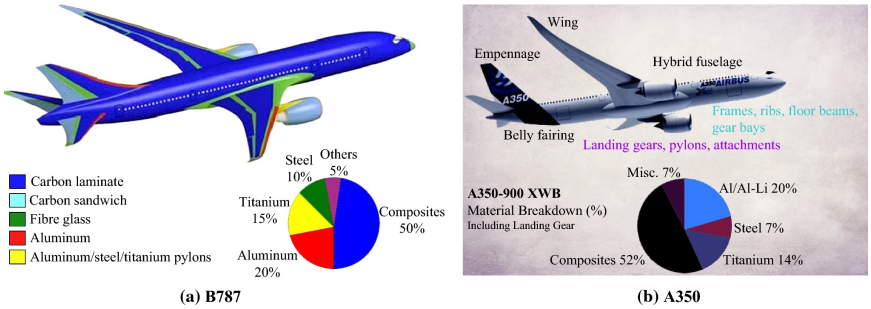


Figure 2.2. The use of composites in aircraft has exceeded 50% such as in the Boeing B787 (50%) and the Airbus A350-900 XWB (52%). [L. Zhang, X. Wang, J. Pei, and Y. Zhou, Review of automated fibre placement and its prospects for advanced composites, Journal of Materials Science, 55 (2020), pp. 7121–7155.]

Fiber reinforced composites not only have the advantages of light weight, high strength, and high temperature resistance, but can also be used to integrate and manufacture large-scale integral components. As opposed to large metal structures which must be fastened together, composites can be laid together into one large piece. Because we are able to create a composition of different material properties, composites can be superior to traditional metal and alloy materials in terms of weight reduction, fatigue resistance, corrosion resistance, reliability, and maintainability, and are becoming more widely used in aerospace, transportation, energy, and defense [34, 40].

In traditional composite fiber manufacturing pre-impregnated sheets of uncured matrix and reinforced fibers are laid down over a form. The fibers are arranged in layers, which are then glued together using either epoxy or polyurethane-based

adhesives. A coating is then often applied for aerodynamics [34]. This has traditionally been done by hand layup.

When designing a structure, the traditional form of fiber composite application is to have the fibers arranged parallel to the direction in which the composite will be stressed the most [34]. When laminated composites are used in applications, different types of fatigue damage can also occur due to off-axis loading with respect to the 0° fiber orientation direction [34]. Due to this, sheet plies are arranged in varying fiber directions in order to provide strength in variable load directions.

Traditionally the desired part is designed and created with Computer Aided Design (CAD) and optimization software. The desired structural properties are obtained by layering thin sheets, or ply layers, in such a way as to provide strength in the response direction. However, increasing the number of plys increases the weight and cost of material used to produce the structure.

Tailored fiber placement techniques involve variable ply fiber directions within a single ply. This allows a single ply to respond to a more complicated load than a unidirectional response, which allows the structure to have the designed stiffness properties with less ply layers. The stacked plys in the center of Figure 2.3 can be replaced with fewer variable tow ply layers.

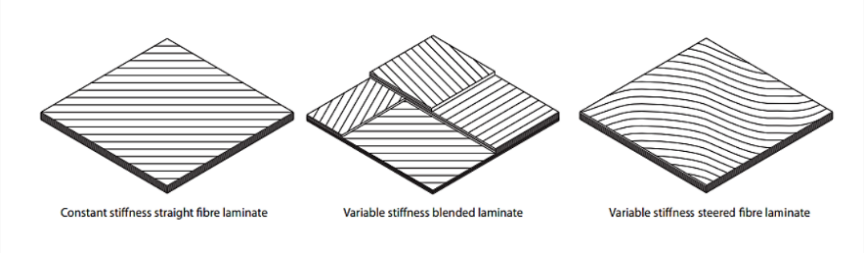


Figure 2.3. Left: Fibers arranged in the direction in which the composite will be stressed the most. Middle: Various plys are arranged in such a way as to handle a more complicated stress profile. Right: A variable tow where the fibers are laid to match the in plane stress profile. [M. Albazzan, Efficient Design Optimization Methodology for Manufacturable Variable Stiffness Laminated Composite Structures, PhD dissertation, University of South Carolina, Mechanical Engineering, 2020.]

The current state of composite research is to design, fabricate, and improve structural performance of advanced composite structures by using tailored fiber

placement [47]. To achieve these performance improvements, the tow steering capabilities of the fiber placement machines are used to align the composite fibers with the desired structural load paths [46]. These curvilinear fiber paths strengthen the structural buckling performance of the composite [40], while reducing the need for more ply layers.

An Automated Fiber Placement (AFP) machine usually consists of a placement head and a robotic arm, or gantry structure. The placement head lays pre-impregnated tows onto a mold to construct the layup. The instructions to the machine are based upon the ideal fiber orientation, which is found through computer optimization software. The process of translating the optimum fiber orientations into viable composite tows is called path planning. Path planning in the AFP process is based principally on analysis or through some algorithm optimization method. The path planning aims to efficiently translate fiber orientations into the physical placement of reinforcing fibers [47]. Researchers have traditionally used genetic algorithms to optimize the fiber placement path [47]. The simulation of the fiber placement process is performed to verify whether the placement direction is accurate and whether the placement path is reasonable and to evaluate the placement ability of different pre-impregnated fiber tows and the accessibility of the placement equipment [47]. Simulation system design is an important part of AFP, which directly affects the efficiency of the placement process and the quality of the fiber-reinforced component [47].

Due to the complexity of the AFP manufacturing process, defects in the pre-impregnated material, and manufacturing errors, the laminates are not exempt from imperfections [46]. These defects are show visually in Figure 2.4 and described here. The imperfections might be gaps and/or overlaps, twisted tows, and fiber waviness, which affect the mechanical performance for the final product [40]. Automated fiber placement machines come with many unintended manufacturing defects, but the most important to this thesis are the gaps and overlaps.

In the process of fiber placement, due to the complexity of the mould surface, the distance between adjacent placement paths cannot be kept constant. When the distance between two adjacent placement paths is not equal to the width of the fiber band, overlaps or gaps of fiber bands easily occur, resulting in partial accumulation or absence of materials [47].

These manufacturing defects that appear in the AFP process can affect the mechanical properties of composites [40]. Marrouze et al. 2013 [25] proposed an advanced computational multi-scale and multi-physics damage tolerance approach in

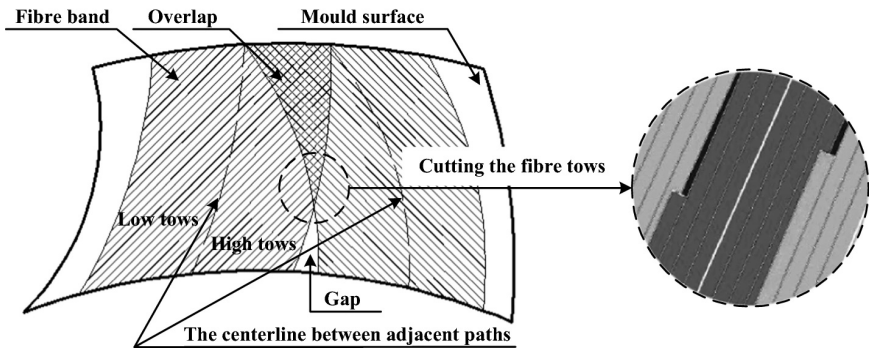


Figure 2.4. An illustration of various manufacturing induced defects. [LL, Zhou Y, Zhou LS (2007) Path planning and wire number solution of composite fiber placement. Acta Aeronaut Astronaut Sin 28(3):745–750]

order to evaluate the effects of defects, which combined micro-mechanics with the finite element method, damage tracking, fracture, and property prediction with and without defects. Research at implementing segments of this process has found that the effect of gaps on in-plane shear properties is more significant than that of overlaps [40].

Our simulation algorithm creates virtual tow paths. This allows us to then diagnose the gaps and overlaps that are created and provide this analysis back to the overall optimization. The details of the simulation are further discussed in Chapter 4.

2.2 Surrogate Models

While parallel research in the group has focused on creating the simulation algorithm, the research presented here focused on creating a surrogate model to replicate the results of the simulation with less computational expense. We used Deep neural networks to replicate the simulation algorithm. Surrogate models rely on samples from the function they wish to approximate. The research area and process of intelligently choosing samples is known as Design of Experiments.

Surrogate models are used to reduce the cost of design and optimization by fitting "approximate" models for complex problems, which are expensive to solve [33]. These surrogate models are useful for design and optimization only if they can predict the underlying behaviour accurately [43]. Surrogate models are also called response surfaces; Kriging, polynomial interpolants, multimodel models and Deep neural networks can all be referred to as response surfaces [4, 33].

No surrogate model is the best choice for every problem domain and the surrogate performance depends on both the nature of the problem and the sampled points [43]. Kriging surrogate models are popular in engineering and design applications but there are drawbacks to Kriging. The form of the Kriging model requires the

inversion of a potentially dense $N \times N$ matrix, where N is the number of sample points. Thus, the basic Kriging method does not scale well for large N. In addition, if two or more of the sample points are close together, the N x N matrix becomes ill conditioned. Thus, while Kriging tends to work well for sparse sets of samples, this method tends to break down as the number of samples increases [41]. This is relevant to our problem as we are going to use our surrogate in an optimization process where we hope that we will be able to perform small perturbations in our design parameters.

Recently, methods based on Neural Networks and, in particular, Deep neural networks have become extremely popular and allow large dimensionality reductions for very high-dimensional, highly nonlinear problems. Neural networks have shown to be effective in a large variety of domains [43].

When the underlying function is well-behaved, lower order models such as Kriging can give a good fit. When the underlying function is highly non-linear, we need a higher order model and consequently higher number of design points [43]. For our problem, the simulation of fiber tow paths is a highly non-linear simulation. We chose to use neural networks, as they perform relatively well and are often chosen for non-linear predictors [23].

Data from the original function or process is necessary in order to create a surrogate model. When it is possible to create samples, the methods used to choose our inputs is known as design of experiments. The main challenge in applying deep neural networks to problems involving physics-based computational models is that they require large amounts of data [43]. When a problem is computationally expensive, the purpose of design of experiments schemes are to choose samples that efficiently sample the design space [43]. These samples are run through the computer simulation. A response surface approximation, or surrogate model, is then generated based on those sample points [41].

The fact that we need a surrogate model is indicative of the fact that the simulation is expensive. We need to balance the cost of creating the data with the accuracy of the model [36]. The number of variables dictates the size of the design space and thus influences the need for a greater number of representative points [43]. Errors in the approximation can be reduced by increasing the data available for approximation, but practically the feasible number of data points is limited by budgetary constraints. For this reason, we need to carefully plan the experiments to maximize the accuracy of surrogates [43].

The most basic option for a Design of Experiments is to choose random inputs to our simulation. Variations of this type of experiment design are referred to as Monte

Carlo sampling. Over a limited number of samples, Monte Carlo schemes do not guarantee full coverage of the design space. Stratified sampling methods are an improvement over random sampling methods because they use a scheme to select points that best cover the design space [6]. The use of space filling designs when sampling deterministic computer experiments creates a data set that treats all of the design space equally [39].

Stratified sampling methods converge much more quickly by subdividing each design variable into N equal probability bins and constraining the sample set to one sample per bin, thus enforcing the exact distribution over the design variable. The effect is a sample distribution proportional to the sample distribution of a converged Monte Carlo analysis. Therefore, these methods are also known as Quasi-Monte Carlo methods [6].

The specific space filing schema we chose was the Latin Hyper-Cube Sampling design (LHS). In LHS design, it is convenient to think of the samples as forming an $n \times k$ matrix of input where the ith row contains specific values of each of the k input variables to be used on the ith run of the computer model [41]. LHS partitions the parameter space, k, into bins of equal probability [41], which selects n different values from each of k variables $X_1, \ldots, X_k$. The range of each variable is divided into n non-overlapping intervals on the basis of equal probability. One value from each interval is selected at random with respect to the probability density in the interval. The n values thus obtained for X_1 are paired in a random manner (equally likely combinations) with the n values of X_2. These n pairs are combined in a random manner with the n values of X_3 to form n triplets, and so on, until n k-tuplets are formed [41]. We have constructed 2 LHS designs in Figure 2.5 to show visually that as we increase the sample size the density of our coverage of the design space increases.

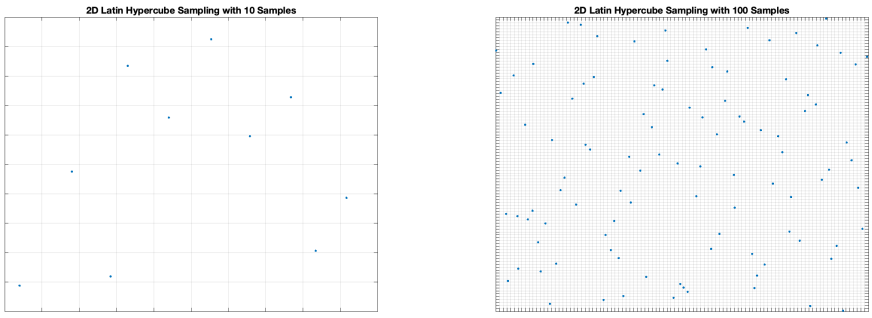


Figure 2.5. Latin Hypercube sampling 2 dimensional design with 10 samples and 100 samples created in MATLAB.

CHAPTER 3
DEEP NEURAL NETWORKS

In this chapter we discuss the fundamental mathematical topics important to construction deep neural networks. This is meant as a brief overview with the goal of explaining some of the more relevant neural network architectures, activation functions, loss and test metrics etc.

3.1 Deep Learning

Deep Learning refers to a subclass of Machine Learning techniques that utilize the neural network learning framework and have many layers of neurons. The motivation behind the field of Machine Learning is that some phenomena in the world, physical, social, political, etc. can not easily be described by a closed form functional representation. However, we do have observations of these phenomena, and the hope is that we can create 'predictors' that if shown enough data will learn to make the predictions with an acceptable level of accuracy. The key aspect of deep learning is that these 'predictors' are not designed by human engineers or mathematicians but are learned from data using a general-purpose learning procedure [23]. Deep learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned [23].

There have been many mathematical schemes and structures proposed as suitable 'predictors.' Popular methods include linear/logistic regression, decision trees, clustering schema, and neural networks. Each method is more suited to certain types of problems depending on the functions that we are trying to learn [17].

Neural networks are very good at discovering intricate structures in high-dimensional data for highly nonlinear problems [23]. The main class of neural network used in supervised learning for classification and regression is the Feed Forward Neural Network also known as a Multi-Layer Perceptron (MLP). Feed Forward Networks have been applied to a wide range of prediction tasks in such diverse fields as speech recognition, financial prediction, image compression, and adaptive industrial

control, and are therefore applicable to many domains of science, business and government [7, 23].

Artificial neural networks (ANN) are parallel, distributed information processing computational models which draw their inspiration from neurons in the brain [7]. This inspiration can be seen visually in Figure 3.1 in how we have constructed the connections between neurons and nodes. In a biological neuron an electro-chemical signal is built up until the the neuron is activated and thus conducts a signal to other neurons it is connected to [28]. A node in a neural network works in a similar fashion in that it is fed input from other nodes, and once the value of the node reaches a certain point it is activated and produces an output through the edges connected to it.

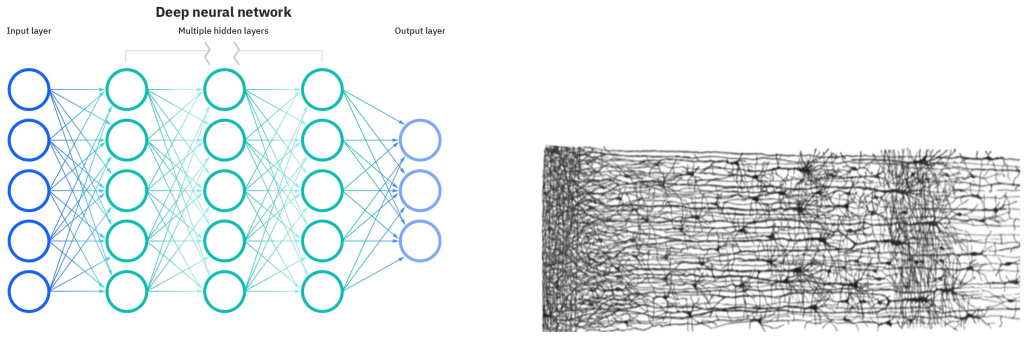


Figure 3.1. Artificial neural networks get their inspiration from the human brain. Left: A symbolic representation of a Deep neural network.[IBM cloud education: What are convolutional neural networks?, Oct 2020.] Right: A rudimentary mapping of a biological neural network. [A. Montesinos Lopez, and J. Crossa, Fundamentals of Artificial Neural Networks and Deep Learning, Springer International Publishing, Cham, 2022, pp. 379–425.]

3.2 Overview of Neural Networks

Neural networks are useful tools to learn behaviors of systems where there is either no functional description or it is unknown. The most basic neural network consists of 3 major sections. The input layer, the perceptron layer, and the output layer. Each layer consists of nodes. These nodes are "connected" to the nodes in the next layer and those connections are given weights. Nodes that are not in the input or output layer are conventionally called hidden units or hidden layers [23]. A Deep neural network is simply a neural network with 2 or more hidden layers.

The neural network makes a prediction by taking in a vector of inputs and then performing a linear combination with a vector of weights to determine the inputs to the nodes in the next layer, Eqn. (3.1).

$$\Theta = \sum w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + ... \tag{3.1}$$

The result of the linear combination is put through an "activation" function [23]. Computing a single layer of nodes can be performed as a matrix vector multiplication. Matrix vector multiplication is a relatively inexpensive operation and thus the forward pass through the network to compute or 'predict' an output is fast. Figure 3.2 allows us to see the flow and transformation of information in the neural network.

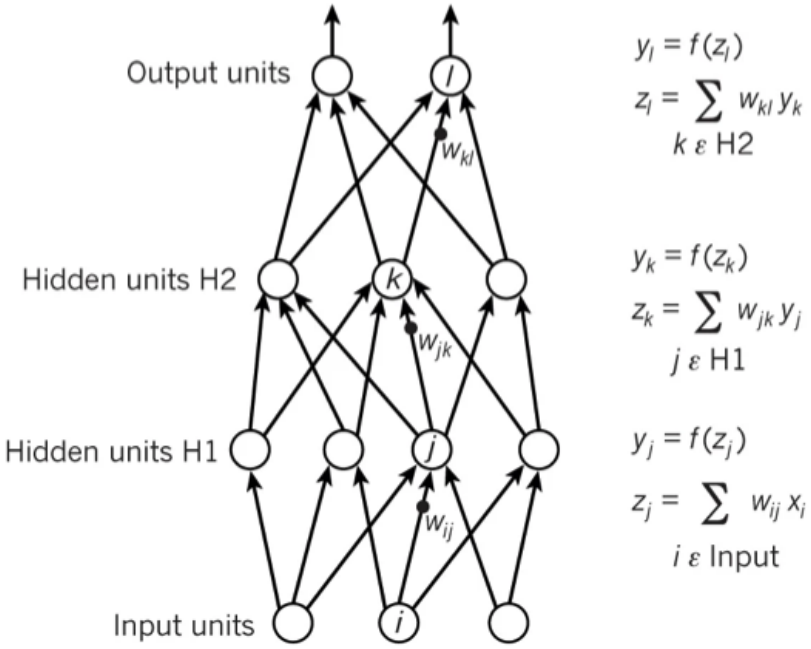


Figure 3.2. A visualization of a the forward pass of a basic neural network. [Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature, 521 (2015), pp. 436–444.]

In the input layer each input variable, along with a extra bias node of value 1, are "connected" to each node in the hidden layer. This connection is the linear combination of the inputs and a weight matrix.

If the input is $X = [x_1, x_2, \ldots, x_n]$, where n is the number of inputs, and the weight matrix is Θ where:

$$\Theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \ldots & \theta_{1h} \\ \theta_{21} & \theta_{22} & \ldots & \ldots \\ \theta_{n1} & \ldots & \ldots & \theta_{nh} \end{bmatrix}$$

where h is the number of nodes in hidden layer, then the input to the hidden layer is defined as $Z_{input} = X * \Theta$. To determine the final values of the hidden layer Z

we pass each node of the hidden layer through an activation function. The archetypal function, Eqn. (3.2), is the Logistic function:

$$Z = \sigma(Z_{in}) = \frac{1}{1 + e^{-Z_{in}}} \tag{3.2}$$

which transforms the input into the range of [0,1] and plotted below in Figure 3.3.

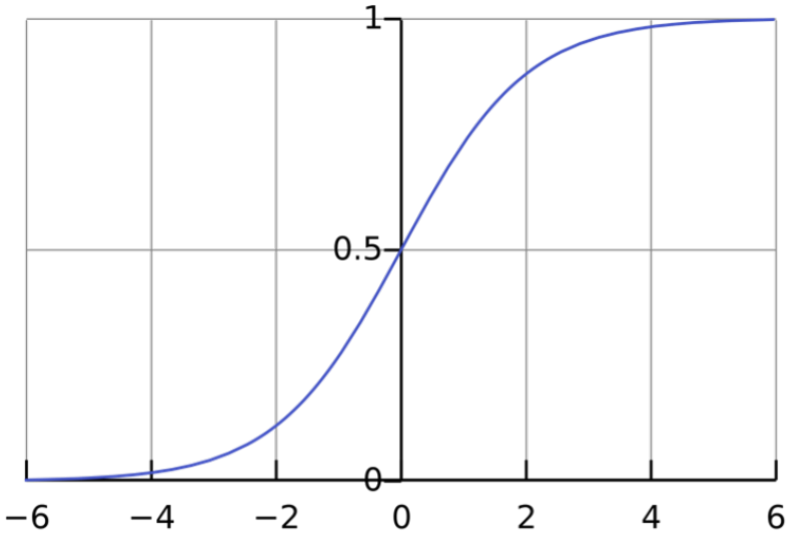


Figure 3.3. A plot of the output range of the Logistic function. Logistic functions also play an important role in Recurrent neural networks, specifically serving as gating functions in Long Short-Term Memory cells.

Once the output of the nodes in the first hidden layer are determined, these are then multiplied by another weight matrix which connects that hidden layer to the next layer. The process is repeated until we determine the value of the output nodes.

In regression problems this output value is taken directly, while in classification problems the output is passed through another activation function. From here a threshold can be set that determines if the final value is 0 or 1, or the value can be left as is and used as a confidence measure.

A non-vectorized representation of the above process is given in Eqns. (3.3) and (3.4).

For a node v in layer l:

$$a_v^l = \sum_{k=1}^{h(l-1)} w_{kv}^l z_k^{l-1} + w_{0v}^l z_0^{l-1} \tag{3.3}$$

$$z_v^l = \sigma(a_v^l) = \frac{1}{1 + e^{-a_v^l}} \tag{3.4}$$

The adjustments of the weights matrices is done during "training" and is accomplished via a process known as back propagation.

3.3 Training The Neural Network

Modern neural networks are trained through a process known as "Back Propagation". The actual mathematical implementation of back propagation in today's industrial Machine Learning libraries is more complicated and efficient than what is presented here, but back propagation is based on the concept of gradient descent.

Neural network training consists of three stages: a forward feed of the input training sample, a calculation and back-propagation of the error, and then adjustment of the associated weights [21]. We determine the error using a loss function that is appropriate for the objective. In classification problems, error accuracy measures are often chosen. For regression problems, variations of the L_1 and L_2 norms like Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) are used.

After the error is computed for the output nodes, the error is propagated back through the network to the previous layer. The basic back-propagation algorithm then adjusts the weights in the steepest descent direction [21]. This process is repeated until the output error is minimized.

From the forward pass presented above, the network produces a prediction $\tilde{y}$. We compute an error from the known y. To simply illustrate the point, we use Squared Error as the lose function, Eqn. (3.5). While there are many schemes to weight initialization, here the weights are initialized randomly.

$$E = \sum (y - \tilde{y})^2 \tag{3.5}$$

In order to update our weights, we use the process of gradient descent and must apply the chain rule, Eqn. (3.6).

$$w_i = w_i - \alpha \frac{\partial E}{\partial w_i} \tag{3.6}$$

Our final output, Eqn. (3.7), is the weighted linear combination of the prior hidden layers' activation nodes and the weight matrix.

$$E = \sum (y - \sum_{k=1}^{h^{(l-1)}} w_{kv}^{l} z_k^{l-1} + w_{0v}^{l} z_0^{l-1})^2 \tag{3.7}$$

Taking the derivative to determine our gradient step yields, Eqn. (3.8):

$$\frac{\partial E}{\partial w_i} = 2\sum(y - \sum_{k=1}^{h(l-1)} w_{kv}^l z_k^{l-1} + w_{0v}^l z_0^{l-1})\frac{\partial a_v^l}{\partial w_i} \tag{3.8}$$

Abstracting the next hidden layer yields Eqn. (3.9)

$$\frac{\partial E}{\partial w_i} = 2\sum(y - \sum_{k=1}^{h(l-1)} w_{kv}^l z_k^{l-1} + w_{0v}^l z_0^{l-1})(-z_i^{l-1}) \tag{3.9}$$

Then we plug in Eqn. (3.9) into Eqn. (3.6) so we can adjust our weights. Propagating backwards to our input layer requires us to determine $\frac{\partial E}{\partial w_i}$ for every hidden layer. Repeated applications of the chain rule are applied.

In this manner, the weights in the network are 'learned.' The entire process is shown in Figure 3.4. As we show the network more data samples, the network performance will gradually improve. The network weights are updated back to front, a layer at a time, until all the weights are updated. This entire process is then repeated for every data sample.

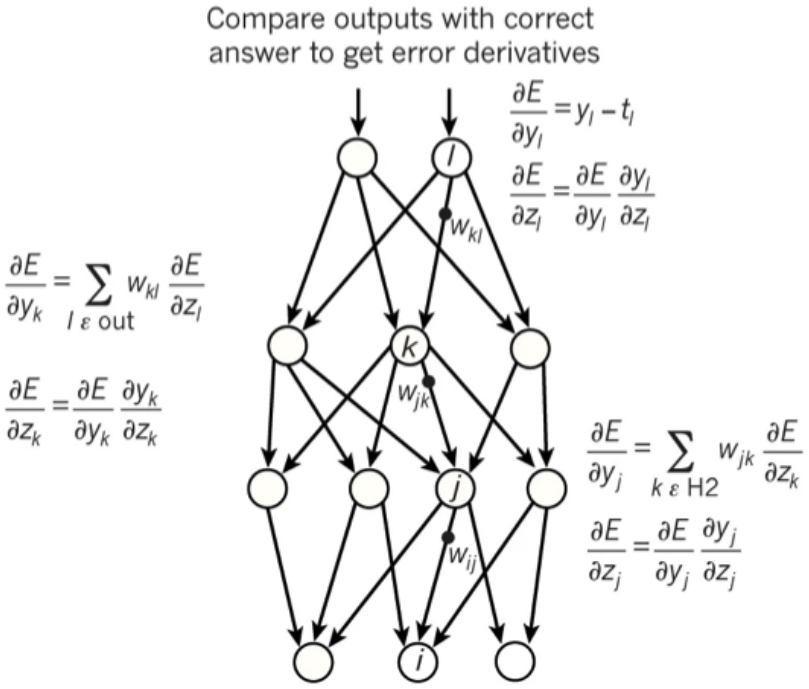


Figure 3.4. A visualization of the backward propagation of error and subsequent weight adjustment in a basic neural network.[Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature, 521 (2015), pp. 436–444]

In practice the data samples are trained in batches and the back propagation only occurs at the end of each batch. Batches are used to manage training time and help the training algorithm from getting stuck in local optima [23].

Iterating through the entire training data set once is called an epoch. After each epoch, the model is used to perform predictions on a validation dataset to test performance of the model on unseen data and gauge the ability of the model to "generalize well." The data scientist predetermines either a set number of epochs or a threshold when validation results are no longer improving to stop the training.

Once the model is trained, it is tested against an independent test data set. Often the training, validation, and test data sets are all randomly chosen subsets of an original dataset. For this thesis, we created 3 independent datasets - each created using a LHS design, but with decreasing number of samples.

The weights in the neural network are also referred to as the parameters. The elements of the neural network architecture; the number of layers, nodes per layer, activation function, loss function etc. are all known as hyper parameters. Choosing and adjusting the hyper parameters, thus designing the neural network architecture, is the core task in engineering machine learning solutions.

3.4 Activation Functions

Activation functions determine how the neurons in the hidden layers are activated. They control the output from one hidden layer's nodes to the next layer. Most importantly, they allow the network to learn non-linearities in the data [23, 28]. While there is research to theorize why certain activation functions work best for certain problems, activation functions are usually chosen because they have been shown to work.

The simplest activation function is the Linear function. $G(z) = \Theta * z$. This used for the output layer in regression problems in order to create a real valued output.

3.4.1 Rectifier Linear Unit

The ReLU activation function, Eqn (3.10), is flat below a certain threshold and then grows linearly, as seen in Figure 3.5. It was first introduced in 1975 [14] but was popularized in Nair and Hinton's 2010 paper [29]. Since the ReLU function is computationally simple, it allows for construction of deeper neural networks as they do not significantly add to training costs.

$$\sigma(x) = max(x, 0) \tag{3.10}$$

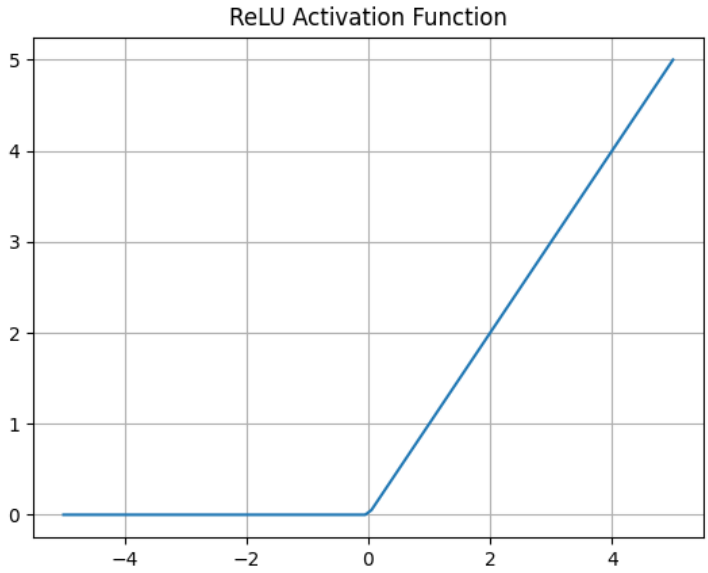


Figure 3.5. ReLU [Pytorch documentation torch.nn]

3.4.2 Leaky Rectifier Linear Unit

ReLU units often suffer from the vanishing gradient problem [28]. A solution to this is the Leaky ReLU unit, Eqn. (3.11). This activation function is a relaxation of the ReLU unit where the function is no longer flat below the threshold, but linear with a small negative slope. While Figure 3.6 may look almost identification to Figure 3.5, this demonstrates that the 'leak' does not dramatically change the behavior of the leaky ReLU from the ReLU activation unit, but is just different enough to avoid the training problems mentioned above.

$$G(z) = \begin{cases} z & \text{if z} > 0 \\ \alpha z & \text{if z} \leq 0 \end{cases} \tag{3.11}$$

3.4.3 Softplus

SoftPlus is a smooth approximation to the ReLU function [12] as can be seen visually when comparing Figure 3.7 to Figure 3.5. The SoftPlus activation function, Eqn. (3.12) has seen more success with regression tasks than with classification tasks. The ReLU activation functions do a better job of creating sparse neural networks, which may help in better performance as classification tasks [16].

$$G(x) = log(1 + e^x) \tag{3.12}$$

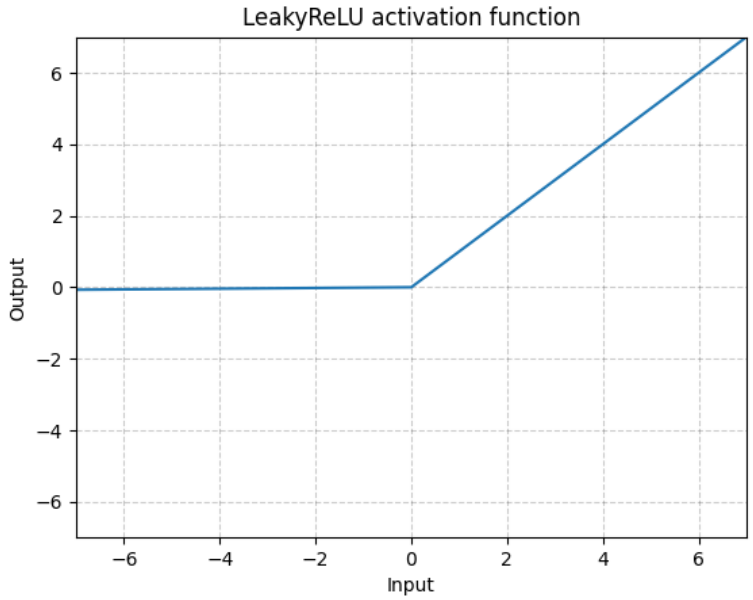


Figure 3.6. Leaky ReLU
[Pytorch documentation torch.nn]

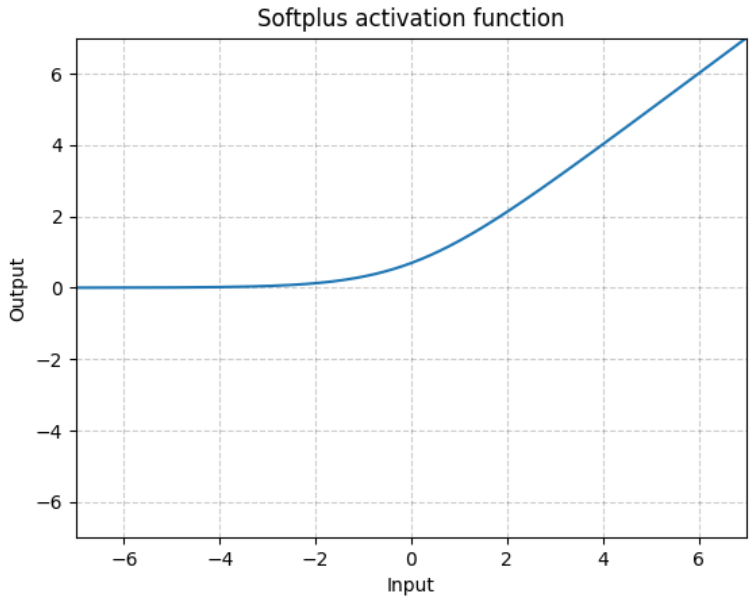


Figure 3.7. SoftPlus
[Pytorch documentation torch.nn]

3.4.4 TANH

The hyperbolic tangent function, Eqn. (3.13), is most popularly seen as the activation functions used in the input layer of Long Short-Term Memory nodes

described below. It also has had success as a general activation function in Recurrent neural networks. This is due to the fact that its output is in the range of $[-1, 1]$, plotted in Figure 3.8 which keeps it from experiencing the exploding gradient problem.

$$G(z) = \frac{e^x - e^{-x}}{e^x - e^x} \tag{3.13}$$

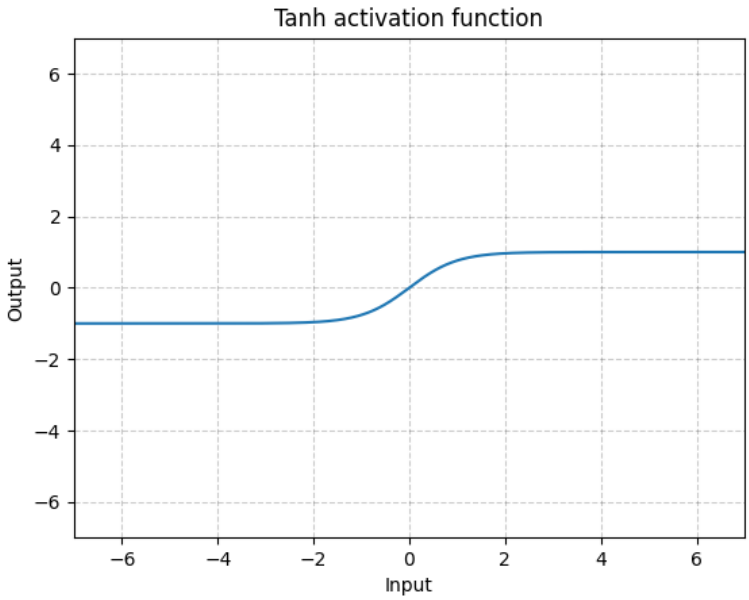


Figure 3.8. TANH
[Pytorch documentation torch.nn]

3.5 Loss Functions and Error Metrics

Loss functions measure the error between the predictions from the ML models and the known output. Due to back propagation's use of gradient descent, they must also be differentiable. This research focuses on a regression task and so we will not talk about loss functions associated with classifications tasks such as cross entropy loss.

3.5.1 Mean Squared Error

The Mean Squared Error, Eqn. (3.14), is the average of the squared error.

$$f(y, \hat{y}) = \frac{1}{m} \sum (y_i - \hat{y_i})^2 \tag{3.14}$$

3.5.2 Mean Absolute Error

The Mean Average Error, Eqn. (3.15), is the average of the absolute error.

$$f(y, \hat{y}) = \frac{1}{m} \sum |y_t - \hat{y_p}| \tag{3.15}$$

Where y_t is our known truth and y_p is our models prediction. The two most commonly used loss functions for deep learning are the Mean Squared Error and Mean Average Error. The Mean Squared error is popularly used because most statistical processes are assumed to have Gaussian distributions and as a rule of thumb Mean Squared Error is more sensitive to outliers than Mean Average Error [10]. Qi et al 2020 [32] showed that MAE would favor Laplacian distributions.

3.6 Testing and Performance Metrics

Once we have completed training the models, we must test their performance. We use a different set of data from the training and validation datasets in order to test the ability of the model to generalize well or measure the performance of the model on unseen data [23]. We also use different metrics to evaluate the model.

3.6.1 Mean Average Percentage Error

Mean Average Percentage Error is a performance metric for regression models, having a very intuitive interpretation in terms of relative error; due to its definition, its use is recommended in tasks where it is more important to be sensitive to relative variations than to absolute variations [10, 11].

Mean Average Percentage Error, Eqn. (3.16), is defined as the error between the true and predicted value scaled by the true value.

$$M = 100 * \frac{1}{n} \sum_{k=1}^{n} \left| \frac{y_p - y_t}{y_t} \right| \tag{3.16}$$

3.6.2 R^2 or Coefficient of Determination

Introduced by Wright (1921) [45] and generally indicated by R^2, the Coefficient of Determination, Eqn. (3.17), original formulation quantifies how much the dependent variable is determined by the independent variables, in terms of proportion of variance [10].

$$R^2 = 1 - \frac{\sum(y_t - y_p)^2}{\sum(y_t - \overline{y_t})^2} \tag{3.17}$$

Where y_t is our known truth, y_p is our model's prediction, and $\overline{y_t}$ is the mean of the known true values. Where The behavior of the coefficient of determination is independent from the linearity of the regression fitting model [10].

The coefficient of determination is defined for the range $(-\infty, 1]$. When $R^2 \geq 0$ it can be interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variables. If $R^2 = 0$: The two numerical variables are independent, that is, they are uncorrelated [10]. Thus, the model performs no better than a guess of the average output of the training data. The coefficient of determination is upper bounded by the value 1, attained for perfect fit. A perfect fit is when all the variance in the dependent variable is explained by the independent variable.

Even if regression analysis has been employed in a huge number of machine learning studies, no consensus has been reached on a single, unified, standard metric to assess the results of the regression itself [10].

While the metrics we use for training and validation are tailored to help the model converge to an optimal solution, the metrics used on the test set are to demonstrate performance. The performance of the model must be communicable to a general audience, especially where we are using a Machine Learning model to solve a real world engineering problem. While MSE and MAE are useful, they do not mean much to those unfamiliar with the problem. Since their values can range between zero and +infinity, a single value of them does not say much about the performance of the regression with respect to the distribution of the ground truth elements [10]. Mean Average Percentage Error and the Coefficient of Determination, R^2 provide better communicability to those who have an understanding of statistics but not specifically neural networks or our aerospace problem domain.

3.7 Machine Learning Architectures

A deep-learning architecture is a multi layer stack of simple modules; all (or most) of which are subject to learning, and many of which compute non-linear input–output mappings [23]. Deep neural networks were chosen due to their ability to

discover non-linear relationships in high dimensional data [23]. Three classes of neural networks were implemented; a Feed Forward neural network (FFNN), a Convolutional neural network (CNN), and a Recurrent neural network (RNN). The following summary is a high level discussion of these Machine Learning Models. A Feed Forward neural network is a general neural network which takes in an array of fixed size and outputs an array of either binary nodes for classification problems or real nodes for regression problems. FFNNs are also referred to as multi-layer perceptrons or often as artificial neural networks, although the latter technically can refer to any neural network. Feed Forward neural networks represent the basic neural network described in the above sections and is visually presented in Figure 3.1. A Convolutional neural network takes a multidimensional array, often an image, and uses multiple convolutional filters to create a feature map, which serves as input to a FFNN. While for this problem the input is not an "image," a CNN was implemented in order to take advantage of the 2-dimensional structured nature of the data. Recurrent neural networks use activation nodes that, in addition to their activation functions, also contain state vectors which store data from past inputs. They are used for sequential and time series problems. A RNN was implemented due to the sequential nature of the underlying simulation algorithm.

3.7.1 Convolutional Neural Networks

Convolutional neural networks are traditionally used for computer vision and image classification [2]. In the description of our introductory artificial neural network described above, our input features are the vector of input values. However, the input layer to an ANN can be more than just the natural input, and could include any number of features comprised of compositions of functions of the original input. When there is already a known relationship between variables in the data, data scientists can create new features from these relationships to input to neural networks. A number of possible features are shown below.

$$X = x_1, x_2, \ldots, x_n, x_1^2, \ldots, x_n^2, \ldots, x_1x_2, \ldots, sin(x_1), \ldots$$

Convolutional neural networks are neural networks that use multidimensional filters to perform convolutions over the input to create a multidimensional feature map. The weights of these filters are trainable and multiple layers can be used to learn complex physical structures. Early convolutional layers may help to detect simple features like edges, while the later convolutional layers help to detect features of the prior feature maps like objects. Multidimensional Convolutions are more useful than FFNNs for images because image data is physically structured data.

The architecture of a typical CNN is structured as a series of stages, show in Figure 3.9. The first few stages are composed of both the convolutional layers already mentioned and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank, show in Figure 3.10.

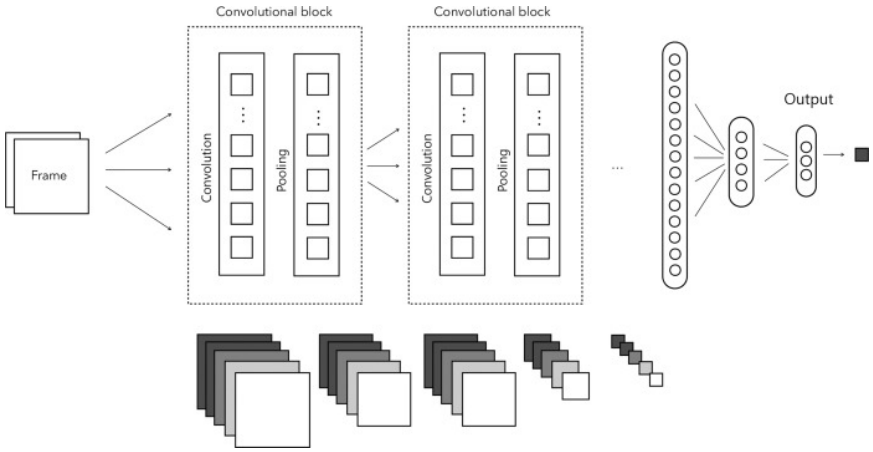


Figure 3.9. Full diagram of a Convolutional neural network. [R. Nanculef, P. Radeva, and S. Balocco, Chapter 9 - training convolutional nets to detect calcified plaque in ivus sequences, in Intravascular Ultrasound, S. Balocco, ed., Elsevier, 2020, pp. 141–158.]

The result of this local weighted sum is then passed through a non-linearity such as a ReLU. All units in a feature map share the same filter bank [23].

The pooling layers can be thought of as features that help to reduce complexity, improve efficiency, and limit risk of over-fitting [2] of the model. There are two main types of pooling: Max pooling and Average pooling. Max pooling is a filter whose output is the maximum input to the filter. Average pooling produces the average of the input values to the filter.

The final segment of a CNN is often standard multiperceptron layers found in the FFNNs. These layers take the information of the extracted features from the Convolutional layers and produce a predictor output value. This may be a probability in a classification problem or a real number for regression.

CNNS are designed to process data that come in the form of multiple arrays; for example, a color image composed of three 2D arrays containing pixel intensities in the three color channels. Many data modalities are in the form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and

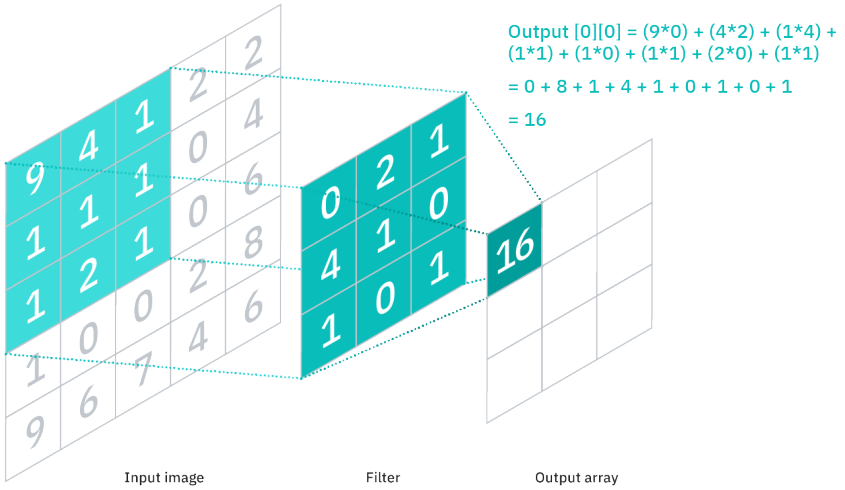


Figure 3.10. Filter performing a simple convolutional of a 2D input. The output is a new feature map and the values of the filter itself are learnable weights. [Ibm cloud education: What are convolutional neural networks?, Oct 2020]

3D for video or volumetric images [23]. CNNs can help the surrogate model capture spatial information inherent in the data.

3.7.2 Recurrent Neural Network

Recurrent neural networks were designed for times series or sequential problems. These deep learning algorithms are commonly used for ordinal or temporal problems [2].

What makes recurrent neural networks unique is that they have mechanisms which act as 'memory.' The structure of recurrent neural networks allows them to place input values in context of the values that came before them. For the basic recurrent neural network this is simply a feedback connection that combines with the input of the next sequential input. Recurrent neural networks were considered since this project is a surrogate of a machine which lays down material in sequential paths.

For tasks that involve sequential inputs, such as speech and language, it is often better to use Recurrent neural networks (RNNs). RNNs process an input sequence one element at a time, maintaining in their hidden units a 'state vector', S_t, that implicitly contains information about the history of all the past elements of the sequence [23].

The original RNNS, however, are more susceptible than other types of DNNs to the vanishing gradient problem [37]. A simplified explanation for this is the fact that the feedback loop, shown in Figure 3.11, weights are applied for every time step, thus amplifying the effect of the weight matrix [19]. Simple RNNs are successful with short term memory, but fail to capture long term dependencies [18].

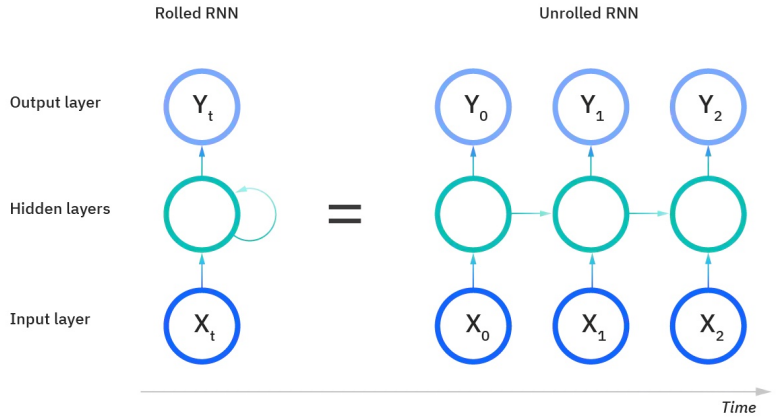


Figure 3.11. Diagram of a Recurrent neural network. Left: A view of the implementation of a RNN with the feedback loop. Right: An 'unrolled' RNN used to demonstrate the behavior and characteristics of an RNN. [Ibm cloud education: What are convolutional neural networks?, Oct 2020]

The prediction from the last time step S_t changes, but the weights which filter to the input of the next time step are the same through out all time steps.

Solving for the exploding gradient is relatively easy. With the use of Sigmoid functions like the logistic or hyperbolic tangent, we can constrain our weights [37]. To correct for vanishing gradients, one idea is to augment the network with an explicit memory. The first proposal of this kind is the Long Short-Term Memory (LSTM) network that uses special hidden units, the natural behaviour of which are to remember inputs for a long time [23].

The LSTM node takes as input the prediction from the last time step, S_{t-1} in Figure 3.12, the input for this time step, X_t, and the memory cell state, C_{t-1}. S_{t-1} is conceptually the short term memory and C_{t-1} is the long term memory.

The first gate, Eqn. (3.18), f_t is a forget gate which helps determine which long term memory is to be forgotten.

$$f_t = \sigma(W_{f1} * X_t + W_{f2} * S_{t-1}) \tag{3.18}$$

where W_{f1} and W_{f2} are weight matrices and σ represents the Logistic function. Since f_t is a vector of values from $[0, 1]$. If any values in f_t are close to 0, this will delete or 'forget' corresponding values in C_{t-1} when we multiply the two vectors.

The input gate, Eqn. (3.19), is similarly defined as the forget gate.

$$i_t = \sigma(W_i 1 * X_t + W_i 2 * S_{t-1}) \tag{3.19}$$

The input gate is used to determine how much of the new information will be included in the updated memory cell state. In Figure 3.12 τ represents the Hyperbolic Tangent function. Our new memory cell state vector is then defined in Eqn. (3.20) as

$$C_t = f_t \bigotimes C_{t-1} \bigoplus i_t \bigotimes tanh(W_n 1 * X_t + W_n 2 * S_{t-1}) \tag{3.20}$$

The output gate, Eqn. (3.21), is defined similarly to the other three gates:

$$o_t = \sigma(W_o 1 * X_t + W_o 2 * S_{t-1}) \tag{3.21}$$

To get our final prediction and the value of the new hidden states, Eqn. (3.22), we perform another multiplication.

$$S_t = o_t \bigotimes tanh(C_t) \tag{3.22}$$

S_t becomes our predictor used for training and testing the RNN and the input S_{t-1} to the next time step.

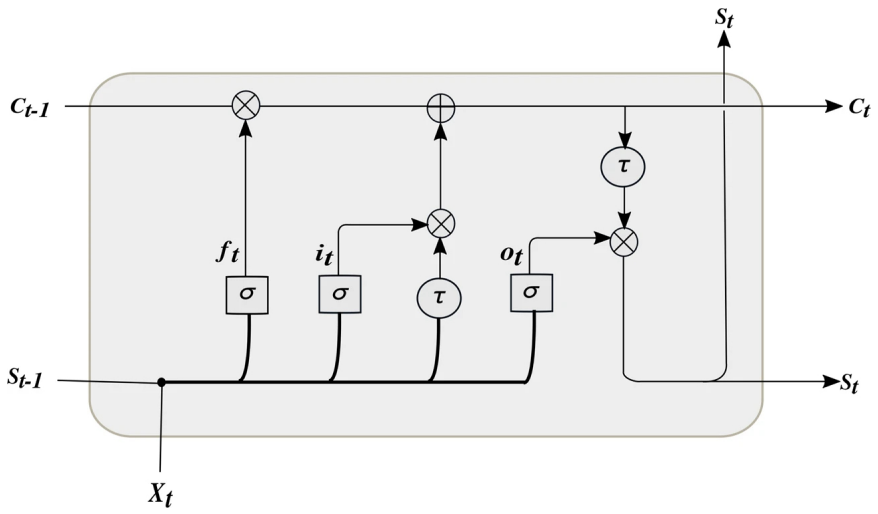


Figure 3.12. Diagram of a Long Short-Term Memory Block. [Sagheer, A., Kotb, M. Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems. Sci Rep 9, 19038 (2019). https://doi.org/10.1038/s41598-019-55320-6]

LSTMs have been shown to avoid the vanishing gradient problem. The composition of the cell state and and how they affect its derivative help prevent the

LSTM gradient from vanishing. The derivative is based upon the inputs from the forget, input, and candidate gates, and the additive and multiplicative combinations of the gates help prevent the gradient from vanishing. These gates help truncate the gradients in order to enforce constant error flow through out the network [18].

LSTM networks have subsequently proved to be more effective than conventional RNNs, especially when they have several layers for each time step [23]. LSTMs have received criticism that they are ad-hoc [19] and their ability to avoid the vanishing gradient problem is not rigorously proven. Regardless, until the recent advent of Transformers, they and their variants such as Gated Recurrent Units have been the go-to DNNs for time series or sequential modeling [42].

CHAPTER 4
DISCUSSION OF SIMULATION MODEL

This thesis creates a DNN that, given a set of ply angles for a coarse mesh on a surface, Figure 4.1, would approximate or predict the gaps and overlap fractional area of that surface. Detailed here is the simulation model that was used to generate the datasets to train the neural networks. This thesis builds on the work of Minaya et al.

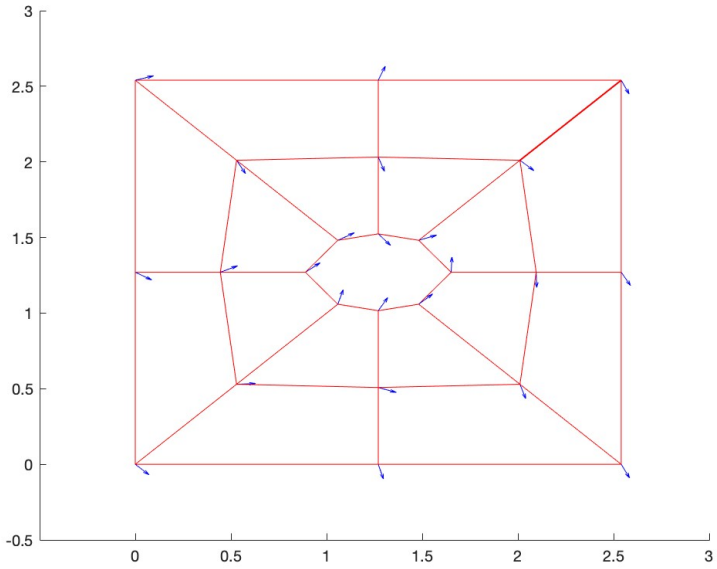


Figure 4.1. Coarse mesh for a flat plate with a hole.

2023 that implemented a path planning algorithm to convert the optimized material angles on a coarse mesh to actual tow paths and use it to quantify overlaps and gaps [27]. The path planning algorithm is the nonlinear function that we are trying to approximate with our DNN. An understanding of the structure of the input data and of the path planning algorithm's logic is an understanding of the problem domain. Certain aspects of the problem domain, detailed below, motivated the neural network architecture design decisions implemented in this thesis.

Several improvements were made to the path planning code to enable it's use for the data creation used to train the DNNs. These included identification of both logical and function errors of the surrogate program and profiling the code to identify and

improve inefficiencies. The final result was a 100 times speed up, which reduced the time to create a data sample from ∼22 seconds to ∼.25 seconds. This made larger datasets feasable by reducing the total computational cost of creating the data samples.

The simulation algorithm can be broken down into 7 main parts. First, the coarse mesh is utilized to interpolate a fine grid mesh. Second, on this fine mesh seed locations are determined and initial fiber paths are created from these seed locations. Third, the fiber tow path is created step by step from the seed location using the fiber orientation mesh to direct the tow. Fourth, the fiber paths are transformed into tow paths by using the manufacturing tow width. Fifth, we determined overlapped tows and large gap areas of the initial coverage map. Sixth, redundant tow segments are cut and gapped areas are filled in with new tows. Finally the overlap and gap fractional area of the final coverage map are calculated. Details can be found in Minaya et al [27].

In both the structural optimization and in creating the tow paths a fine mesh is created by interpolation from the material orientations specified at the nodes of the coarse mesh. Figure 4.2 shows a fine mesh in dark blue created from the coarse mesh in light blue. Both a fine angle mesh and a fine resolution of the surface are used to create a more accurate representation of the tow path reconstruction. In the earlier iterations of the simulation program which generated the flat plate data, this interpolation was done in MATLAB. In the later iterations of the code, the fine and coarse nodal angles and connectivity were provided from the larger optimization scheme to the program.

To create the tow path trajectories a 'Eularian approach' was used. A seeding location is determined. This process is briefly described here. From this seeding point the local fiber orientation is determined by interpolation. The fiber orientation is then used to advance the tow a small increment, proportional to half the tow width. At this new point the fiber orientation is determined again and the tow is advanced in that direction. This process is repeated until the fiber path reaches a boundary. For the next tow path, a new seed location is determined and a new fiber path is traced out. In Figure 4.3 two sample fiber paths are shown in the lower left segment of the surface. Once the initial set of fiber paths is determined, they must be qualified.

The initial fiber paths are laid over a fine reconstruction mesh of elements. The algorithm then traces along the fiber paths, determining which elements are with in a radial distance from the center path line. If the centroid of an element lies within the radius, then the element is marked as covered. The radius is determined from the tow width of the simulated machine. The colored circles in Figure 4.4 are a visual representation of the radii being use to determine elements covered by the tow path.

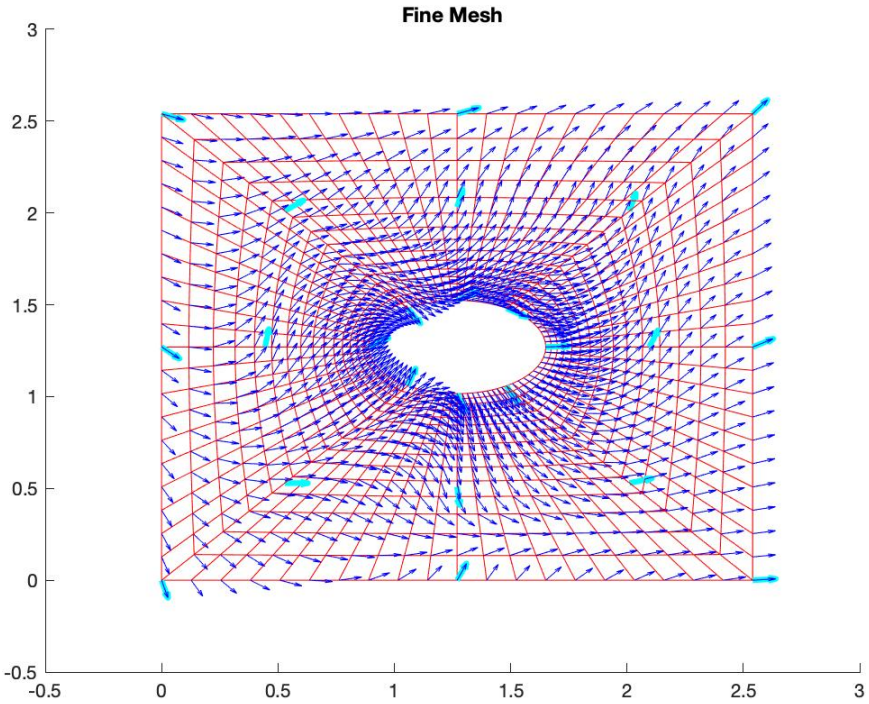


Figure 4.2. The fine mesh is interpolated from the coarse mesh and used for the simulation algorithm.

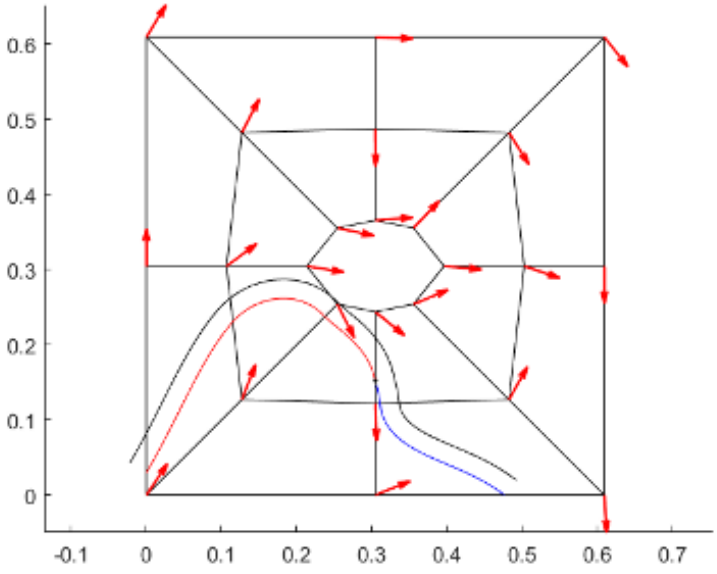


Figure 4.3. The fine mesh is interpolated from the coarse mesh.

Elements store their thickness values. Zero indicates no coverage or a gap. One indicates a single tow or even coverage. Two or more indicate overlapping coverage.

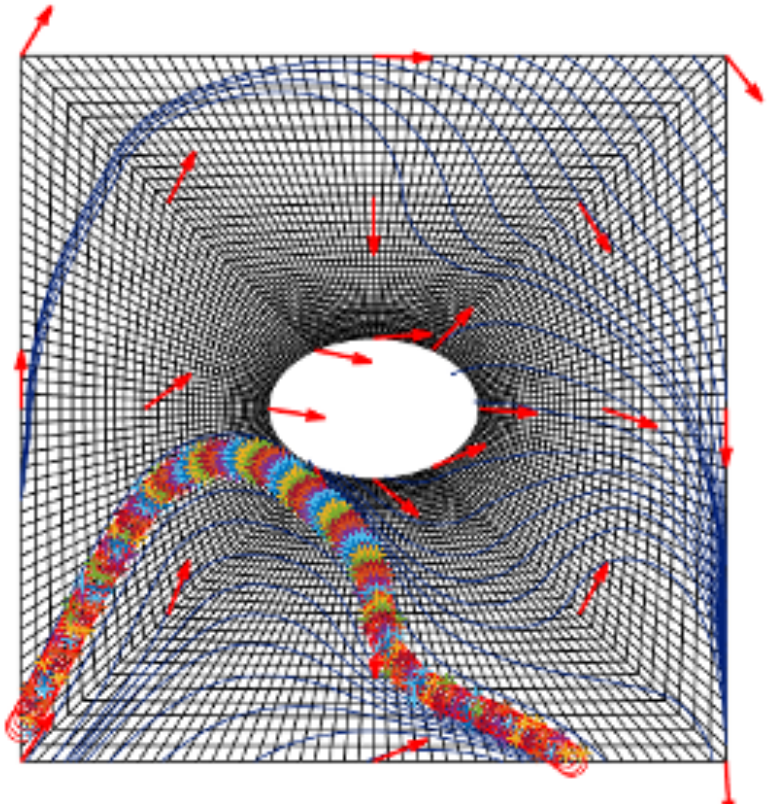


Figure 4.4. The tow width is used to as the radius of a circle whose center follows the tow path. Elements that lie within this circle are considered covered.

Once all fiber paths have been traced out and a fill array is formed, which this is plotted as a heat map/contour plot in Figure 4.5, is analyzed to determine gap areas and regions of overlap.

Regions of high overlap are identified and redundant fiber paths are deleted. Once the redundant tows are cut, regions of gaps are identified for a second fill. The end result of this step is shown in Figure 4.6. This search step adds to the computational expense of the program. A Double Connected Edge List (DCEL) data structure was used to logically highlight regions of gaps. The highlighted regions are then infilled with tows, Figure 4.7 and the process is repeated until some threshold of gap areas is met. The DCEL allows for a constant timed search computation with the trade off of a fixed increase in memory cost.

The final heat map, Figure 4.8 is obtained and the gap overlap metrics, Eqns. (4.1) and (4.2), are computed according to equations:

$$A_O = \frac{\sum_e (-(t_e - 1)A_e)}{\sum A_e} \tag{4.1}$$

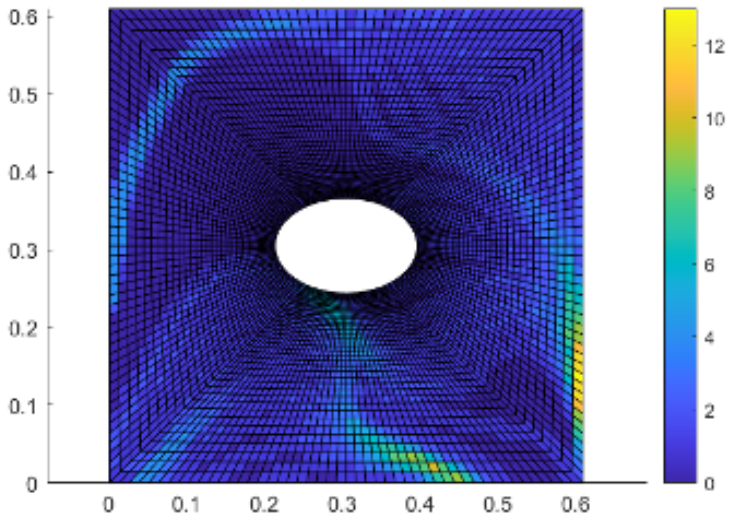


Figure 4.5. An initial heat map after a first run of the algorithm.

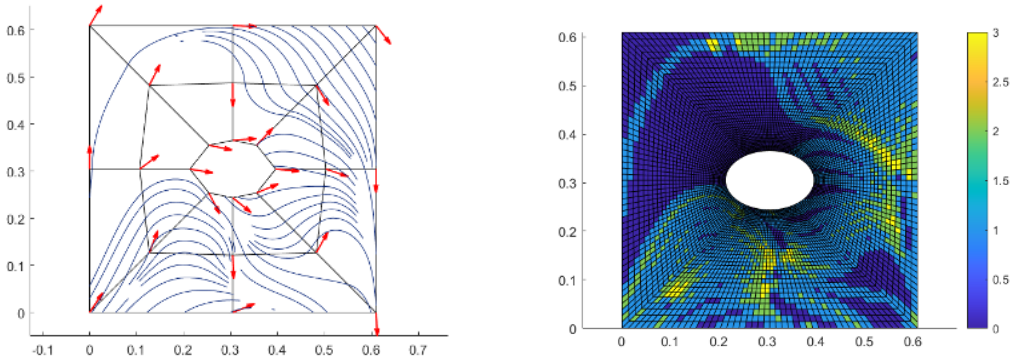


Figure 4.6. Left: Overlapping fibers are identified and cut. Right: A heat map is created and analyzed for large gaps after overlapping tows are cut.

$$A_G = \frac{\sum_e ((t_e - 1)A_e)}{\sum A_e} \tag{4.2}$$

where t_e is the thickness of the element and A_e is the area. These metrics are returned to the overall optimization function.

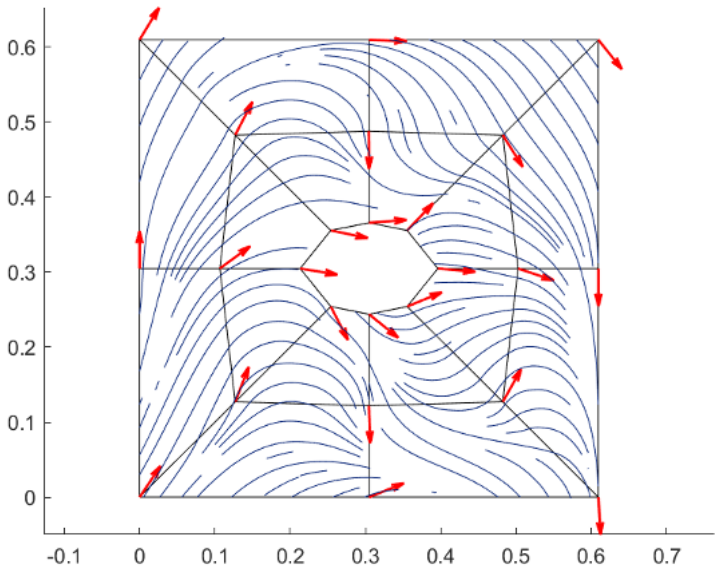


Figure 4.7. New tows are laid to fill in gaps.

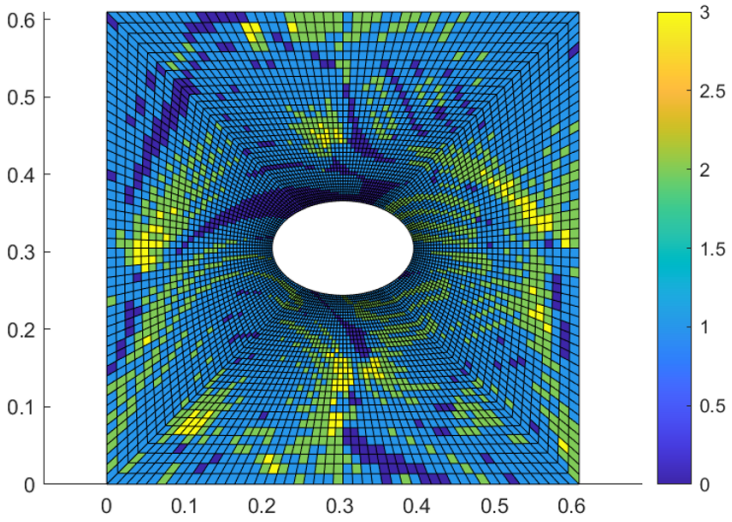


Figure 4.8. A final heat map of tow coverage is created.

CHAPTER 5
METHODOLOGY

This chapter describes the methodology and workflow of this Thesis project. The processes chosen and the motivations behind design decisions are described. The work of this thesis can be split into two halves. The first is using the simulation algorithm to create useful datasets. The second is designing neural network models that accurately represent the original simulation.

While it is useful to conceptually think of creating the datasets and using those datasets to create deep neural networks as two separate tasks, the particulars of each task influenced and provided feedback to the other. Performance and functional analysis of the deep neural networks led to improvements of the simulation algorithm and development of better data. Inherent characteristics of the simulation algorithm determined the structure and computational cost of the data, which dictated and inspired the neural network architectures.

There were also design considerations from the larger structural optimization program that affected the simulation algorithm, which again affected the creation of the datasets. This is most important for the second problem domain described below. While the technical details of the surrogate and larger optimization problem affected the workflow of creating the datasets, the "physics" of the larger problem were the motivation for different network architectures.

5.1 Problem Definition

Before discussing the datasets, it is useful to describe the two problem domains that were covered in this research. Both problem domains were run through the same simulation algorithm described above. The input to the simulation function was the coarse ply angles and geometric information of the surface. The output of the simulation function was a heat map of the surface describing the coverage information. This thickness map was then analyzed to quantify and distill the gap and overlap information into two metrics representing the fractional overlap area and gap area of the surface, defined by Eqn. (5.1)

$$A_G = \frac{\sum(Elem_{thickness<1} * ElemArea)}{TotalArea} \; A_O = \frac{\sum(Elem_{thickness>1} * ElemArea)}{TotalArea} \quad (5.1)$$

The simulation algorithm requires geometric information of the surface in order to construct the surface and then perform its simulation of the tow paths of an AFP

machine. This geometric information includes the nodal location and nodal connectivity information. The neural networks are only being trained to make predictions on a fixed surface for each model. That is, the geometric information does not change and the geometric information, while an input to the simulation algorithm, is not an input feature to the DNNs. During the overall optimization, only the control angles at the ply level are adjusted and so only those angles are relevant to the neural networks.

5.1.1 Problem 1: Flat Plate

Our first problem is a flat plate with 16 control angles, as shown in Figure 5.1. For the design each nodal angle can be varied independently. The input can be represented as a vector of 16 angles, Eqn. (5.2):

$$D_{in} = [\phi_1, \phi_2, \ldots, \phi_{n=16}]. \tag{5.2}$$

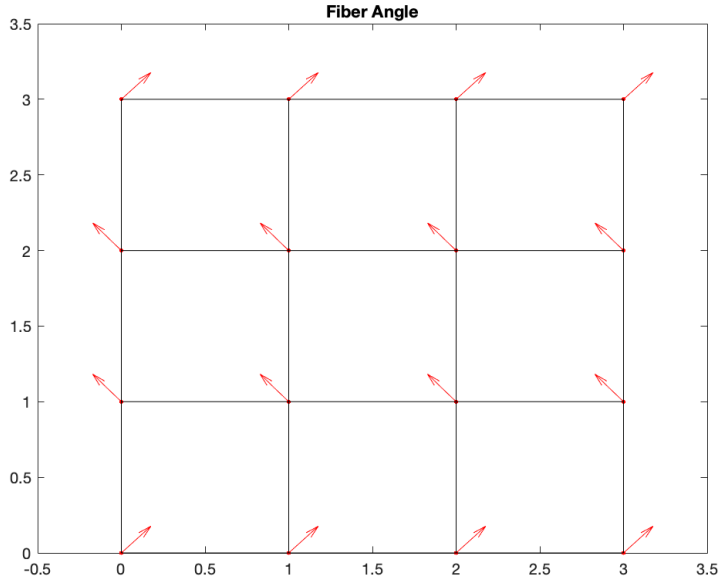


Figure 5.1. Problem 1. A flat plate with 16 coarse angles per ply.

5.1.2 Problem 2: Flat Plate with a Hole

Our second problem domain is a flat plate with a hole in the center. This problem is used to illustrate how the simulation algorithm works in Chapter 4. The input is illustrated in Figure 4.1. For this problem there are 24 coarse angles, Eqn. (5.3).

$$D_{in} = [\phi_1, \phi_2, \ldots, \phi_{n=24}]. \tag{5.3}$$

The output is still the same two metrics $[A_G, A_O]$.

5.2 Creating Datasets

The original simulation algorithm was designed as a project in MATLAB, which took full advantage of the development aids provided by MATLAB. In order to create the datasets, first we needed to create a wrapper function for the simulation code. Due to coding constraints of engineering applications used in the overall optimization structure, the code also had to be coalesced into a single script.

5.2.1 Input

The input to the simulation function was the coarse ply angles and geometric information of the surface. The output of the simulation function was the gap and overlap fractional area of the surface. In Brooks et al [8] they discuss the connection between gaps and overlaps, and the curl and divergence of the vector field composed by the ply angles. We used their results as inspiration and shape the input into a matrix, Eqn. (5.4), and treat it as a vector field:

$$\begin{matrix} \phi_{1,1} & \phi_{1,2} & \phi_{1,3} & \phi_{1,4} \\ \phi_{2,1} & \phi_{2,2} & \phi_{2,3} & \phi_{2,4} \\ \phi_{3,1} & \phi_{3,2} & \phi_{3,3} & \phi_{3,4} \\ \phi_{4,1} & \phi_{4,2} & \phi_{4,3} & \phi_{4,4} \end{matrix} \tag{5.4}$$

and compute the curl and divergence of the input. For Problem 1, these values are relatively computationally inexpensive to compute using built-in MATLAB functions. They are folded into the input of the data sample so that each data sample in the final dataset has the form of Eqn. (5.5).

$$D_{in} = [\phi_{1,1}, div_{1,1}, curl_{1,1}, \phi_{1,2}, div_{1,2}, curl_{1,2}, \ldots, \phi_{n,n}, div_{n,n}, curl_{n,n}, A_G, A_O]. \tag{5.5}$$

where (n, n) is the designated coarse mesh node. The rationale for including the divergence and curl field data is to provide the Machine Learning model with extra features that contain positional and physical data between the input angles. For Problem 1, we were able to accomplish better results with these features and the final models discussed in Chapter 6 include these features.

The ply angles in Problem 2 are arranged in a non-uniform grid. Computing the divergence and curl of a non-uniform grid is more computationally expensive than that

of a uniform rectangular grid. Due to this increase in computational cost we did not include these features in most of the datasets for Problem 2. This will be shown in Chapter 6.

5.2.2 Efficient Data Sampling

Due to the large design space it is important to try and create as large of a dataset as possible to train the neural networks. However, generating the training data was computational expensive. The initial simulation code took approximately 22.5 seconds to compute one sample. Monte Carlo methods require a large amount of samples to fully sample the design space. A LHS design was used in order to create adequately large datasets while evenly sampling the design space.

Three datasets of size 500,000, 60,000, and 10,000 were created, each with their own independent Latin Hypercube Samplings of the design space. Physical considerations of the AFP machine manufacturing process allowed us to limit the design space and ply angles were limited to 0° to 180° for Problem 1 and -90° to 90° for Problem 2.

5.2.3 Technical Considerations

Creating a set of inputs to run through the simulation algorithm in order to train the Machine Learning models also served as the testing and refinement phase of the simulation algorithm. In order to decrease the time to create a sample, I profiled the simulation code and was able to identify inefficient functions and subroutines. The majority of these inefficiencies were caused by calls to generic function solvers which could be replaced with hard coded solutions, nested within if-else statements. While this increased the lines of code, the compute time for one sample was decreased to approximately .225 seconds. The LHS designs also created data samples which exposed logical and coding bugs in the simulation algorithm

The datasets were created via MATLAB scripts but these were run in the command line utilizing the '-batch' flag. This allows MATLAB to run and compile MATLAB scripts without running the Java Virtual Machine, which is important as working with large datasets in the MATLAB JVM is extremely slow due to memory management issues. For Problem 2, this was not possible due to coding considerations and so the input and output dataset files had to be split up and combined before constructing the final datasets required for training the neural network.

5.3 Creating Surrogate Models

In this section the different architectures of the surrogate models are described, referencing back to the principles from Chapter 3. WE then describe the process for training these neural networks.

5.3.1 Creating the Machine Learning Models

Due to the novelty of this application, specific research was unavailable to build upon for designing neural networks to surrogate the AFP process. Trial and error as well as using knowledge of the physical process used to create the data were most important to choosing the final neural network architectures.

The machine learning models were created using the Tensorflow library in Python. The data was read in and separated into input and output. The data was then normalized in order to help convergence during training.

The input data for the CNN was shaped to the two dimensional form seen in Eqn. (5.4). The FFNN and RNN were left as 1D vectors. For Problem 1, all 3 inputs have three channels for angle, divergence, and curl respectively. For Problem 2, there is only a single channel for the angles.

Then the neural network models are defined and compiled. The architecture of the deep neural network is a description of the number of layers, number of nodes in each layer, and the activation functions used in each layer. Adjusting and choosing the right combination is referred to as tuning the "Hyper Parameters."

For Problem 1, the architectures and hyper parameters were found by experimentation by hand using inspiration from work by [24] and [44]. For Problem 2, a hyper parameter search was conducted with the KerasTuner library, but the search did not return better models then those found by experimentation.

The final architecture of each deep neural network for Problem 1 is given in Table 5.3.1 below. The final architecture of each deep neural network for Problem 1 is given in Table 5.3.2 below. The initial models were developed factoring only the overlap value. All 3 networks were trained utilizing the ADAM optimizer method [22], an alternative method to stochastic gradient descent, with a learning rate of 0.01. The loss function for the feed forward and convolutional neural networks was MAPE. The loss function for the recurrent neural network was MAE.

Table 5.1. Neural Network Architectures and Parameters

Feed Forward Network	Convolutional Network	Recurrent Network
Layer 1: 16 Input Nodes	Layer 1: Convolutional 2D: Filters = 16, Kernel Size = (1,1,3) Stride = 1, Activation Nodes = RELU	Layer 1: 64 LSTM
Layer 2: 16 Softplus	Layer 2: Batch Normalization	Layer 2: 64 LSTM
Layer 3: 32 Leaky RELU	Layer 3: Average Pooling 3D: Size = (2, 2,1), Strides = 1	Layer 3: 64 Leaky RELU
Layer 4: 32 Leaky RELU	Layer 4: Convolutional 2D: Filters = 16, Kernel Size = (2,2,1) Stride = 1, Activation Nodes = RELU	Layer 4: 64 Leaky RELU
Layer 5: 32 Leaky RELU	Layer 5: Batch Normalization **Flatten**	Layer 5: 1 Linear
Layer 6: 16 Softplus	Layer 6: 64 Soft Plus	-
Layer 7: 1 Linear	Layer 7: 64 Soft Plus	-
-	Layer 8: 32 Leaky RELU activation nodes	-
-	Layer 9: 1 Linear output node	-

5.3.2 Training Deep Neural Networks

The models for Problem 1 were trained for 70 epochs. A plot of our training performance per epoch in Figure 5.2 shows that the models appear to be converging towards a solution.

For Problem 2, both models were trained for 100 epochs. Mean Squared Error was chosen as the loss function, Mean Average Percentage Error was used for the validation metric. A plot of our training performance per epoch in Figure 5.3 shows that the models appear to be converging towards a solution.

Table 5.2. Neural Network Architectures and Parameters for Problem 2

Feed Forward Network	Recurrent Network
Layer 1: 24 Input Nodes	Layer 1: 64 LSTM
Layer 2: 48 RELU	Layer 2: 64 LSTM
Layers 3-18: 84 Leaky RELU	Layer 3: 64 Leaky RELU
Layer 19: 24 Leaky RELU	Layer 4: 64 Leaky RELU
Layer 20: 1 Linear	Layer 5: 1 Linear

In further testing, the RNN was quickly over trained. At 137 epochs the validation results became worse than the training results. The MAPE worsened from 18% to 23% on the validation dataset between the 136 and 137 epochs, and the testing R^2 and MAPE values computed for the test data were also degraded. This suggests that the model is too simple for this problem.

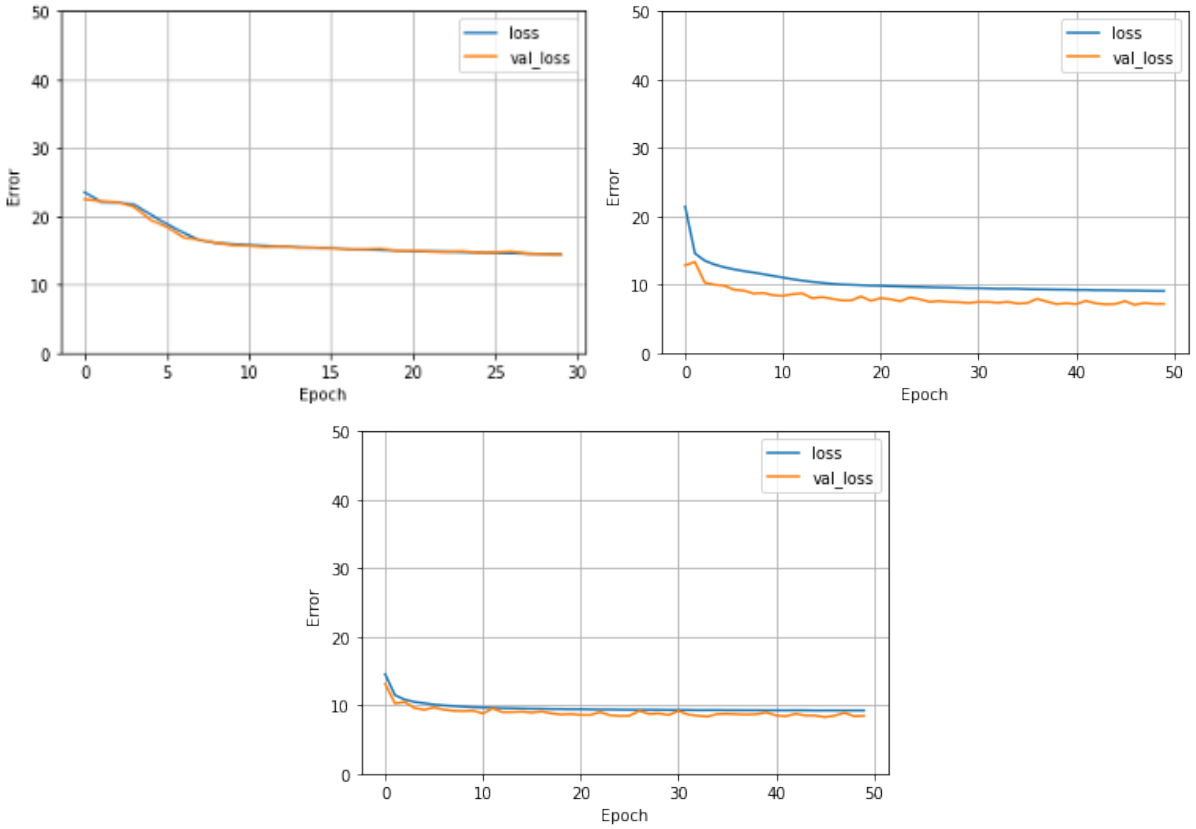


Figure 5.2. The training plots for the 3 DNN models for Problem 1. Top Left: Feed forward neural network. Top Right: Convolutional neural network. Bottom: Recurrent neural network.

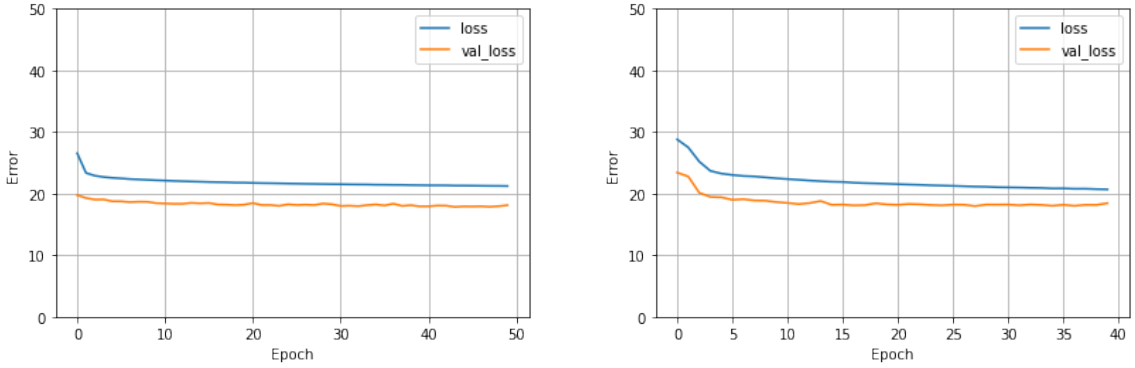


Figure 5.3. The training history of the first 50 epochs of the Feed-forward neural network (left) and the Recurrent neural network (right) on Problem 2, the hole in the plate problem.

CHAPTER 6
TESTING AND RESULTS

6.1 Testing

The neural networks were trained, validated, and tested with 3 datasets generated using 3 independent LHS design distributions with sample sizes of 500,000, 60,0000, and 10,000, respectively. The predicted accuracy of the fitted models at the test sample points (10,000) are quantified by the Mean Absolute Percent Error and the Coefficient of Determination, R^2.

6.1.1 Flat Plate

In the first problem domain, the dataset representing a flat plate with 16 control points arranged in a 4 by 4 grid is enhanced by including the local divergence and curl at those points. Thus the input is 3 channels deep and layered with the divergence and curl in the second and third channels. This inclusion of divergence and curl is discussed above in Chapter 5 and is based upon the work of Brooks and Martin [8].

Our results for the LHS 10,000 sample test set are presented in Table 6.1. The neural network models are trained to predict the percentage overlap of the simulated surface. The final models have the architecture and hyper parameters detailed above in Table 5.3.1.

Table 6.1. Neural Network Test Data Performance Results

	'Known True' Average Wall Clock Time Per Sample	Mean Absolute Percentage Error	R^2	'ML Prediction' Average Wall Clock Time Per Sample
Feed Forward Network	.225 Secs	10.508%	0.72470131	2.362×10^{-5} Secs
Convolutional Network	.225 Secs	8.103%	0.81009918	1.329×10^{-4} Secs
Recurrent Network	.225 Secs	6.225%	0.88566867	7.869×10^{-5} Secs

From Table 6.1 we can see that the recurrent neural network performed the best on the test data set in terms of MAPE and R^2. Since the speed up achieved over the simulation algorithms by the neural networks models is much greater than the relative difference between the models, we will only note that the Convolutional Network was the slowest of the three.

Figures 6.1, 6.2, and 6.3 are scatter plots of true versus predicted values of the overlaps on the surface. A reference line with slope 1 is plotted to orient the data.

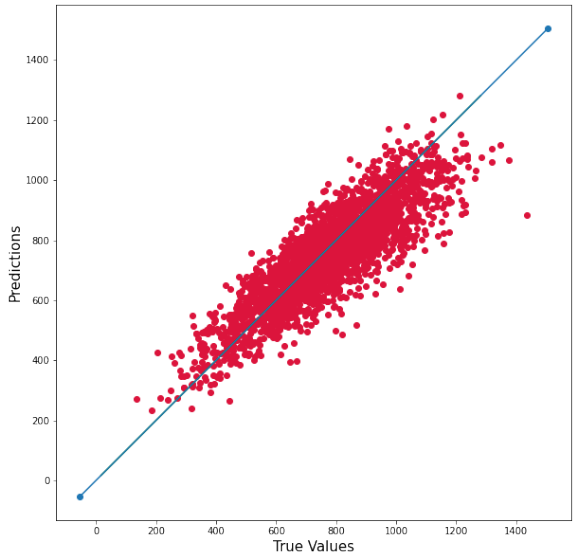


Figure 6.1. A scatter plot of predicted versus true value for the feed-forward neural network. This type of chart can be used to visually compare model architecture performance and to identify regions where the model is not as accurate.

6.1.1.1 Test 1

The raw metrics are useful for determining how effective a model is compared to other models with different hyper parameters or between model types such as RNN vs CNN. While these metrics give a quantitative indication of their usefulness as a surrogate model, we construct specific test cases to create a qualitative analysis of their usefulness.

Visualizing high dimensional design spaces is difficult. To visualize how the overlap functionally varies with material orientation angles, we performed 1 dimensional explorations of the design space and compared the performance of the three models. In Figure 6.4 we present a constructed example where we hold the top and bottom rows of angles fixed at a 45° and rotate the middle two subsets of angles from the horizontal at 0° to 180°.

We expect to see 0 overlap and gappage area when the test angles match the fixed angles. That is when all angles are aligned equally at 45°. As the test angles move

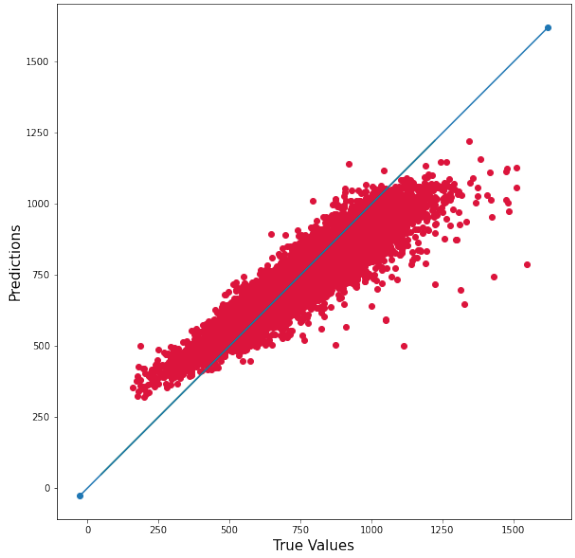


Figure 6.2. A scatter plot of predicted versus true value for Convolutional neural network.

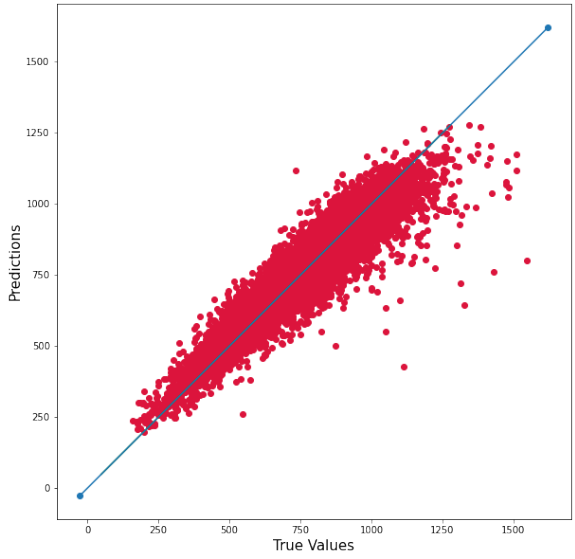


Figure 6.3. A scatter plot of predicted versus true value for Recurrent neural network.

towards and away from this alignment we would expect that the induced errors would subside and increase appropriately. This test case is illustrated in Figure 6.4

The functional results for the 3 neural networks are illustrated in Figure 6.5. We can see that all of the models perform relatively poorly when the true overlap and gap values are lower. These are situations when all of the ply angles are oriented in the

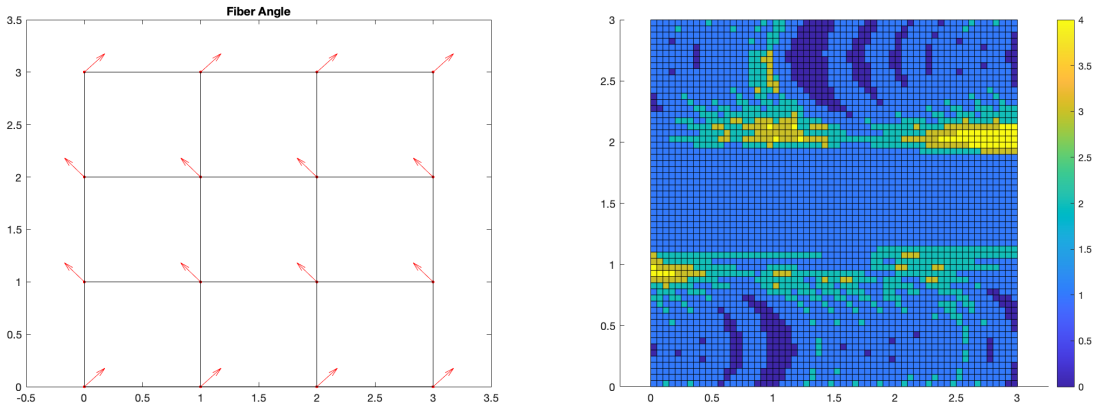


Figure 6.4. On the left is a visual presentation of input for the first test case. In the first test case the middle rows are rotated from 0° to 180° while the top and bottom rows are held at 45°. The snapshot is when the middle rows are at 135°. On the right is the heat map used to to display the overlap and gap information and to calculate the overall fractional area of gap and overlap.

same direction. This can also be seen when examining the red scatter plots in Figures 6.1, 6.2, and 6.3. The axis scale represents the raw overlap element count. Seeing that there are few samples below 250, we normalized 250 by the total element count of 3600. which comes to 6.9 percent overlap coverage. We can see on the vertical Axis of Figure 6.5 that this is exactly where the models start to fail. The LHS design is a space filling method, but when considering all the possible ply angle orientations, situations where even coverage of the surface and thus low number of overlap and gaps occur are relatively rare. For our application this represents a drawback to the Design of Experiments technique.

In an attempt to improve the predictions in this lower range, we filtered a LHS dataset with 2 million samples by removing all samples that had any set of angles that were greater than a 45° offset with any one of their immediate neighbors. The 2 million sample dataset was filtered down to just 4226 samples, representing just .2% of the original dataset. Retraining the network on this dataset worsened the results of the model to a MAPE of 11.18% and an R^2 0f .80. Ultimately we chose another DOE scheme Box-Behnken Design to provide the neural networks with more data in this region. We will discuss this below.

While the models perform poorly when the overlap area is less than approximately 7%, in Figure 6.5 the performance above this region does show a model which closely represents the functional behavior of the original simulation. These models were trained on a LHS design distribution of samples, but this specific test

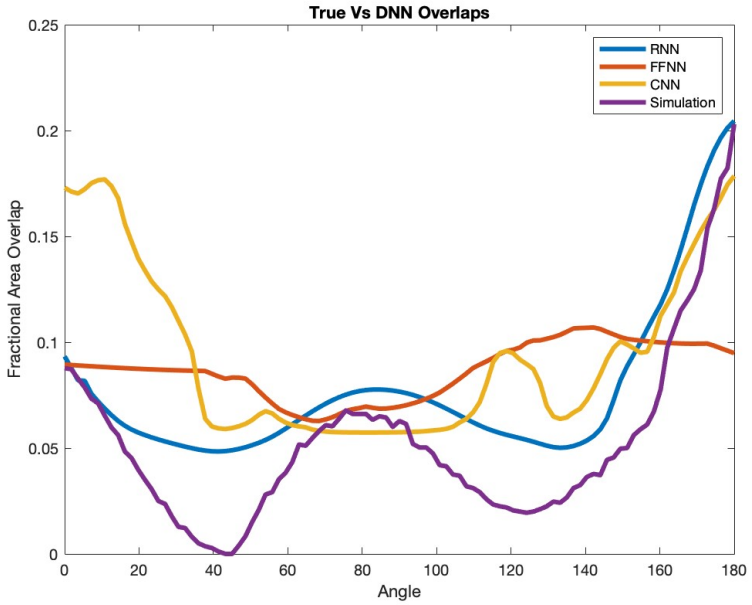


Figure 6.5. A functional test of the performance of the feed-forward, convolutional, and recurrent neural networks for the test case presented in Figure 6.4

shows that the model is acting as a functional representation of the system. As was the case when comparing the R^2 and MAPE metrics, from visual inspection of Figure 6.5 the model performance of the recurrent neural network is better than the convolutional neural network, which is better than the feed-forward neural network.

6.1.1.2 Test 2

For the second test case we rotate the upper right and lower left center angles, diagonally opposite each other, from 0° to 180° while holding all other angles fixed at 45°. The test case is visually presented in Figure 6.6 and the results are presented in Figure 6.7. In this test case the RNN and CNN model performance is visually comparable. Both CNN and RNN model plotted predictions align well with the simulation algorithm well. We see again that all three neural networks can not predict the behavior below approximately 7 percent.

6.1.2 Flat Plate With a Hole

For the second problem of a flat plate with a hole in the middle, we created a feed-forward neural network and a recurrent neural network. This input is a non-uniform grid of 24 control angles that have an input range of -90° to 90° with 0° being the right horizontal. Again the neural networks were trained, validated, and

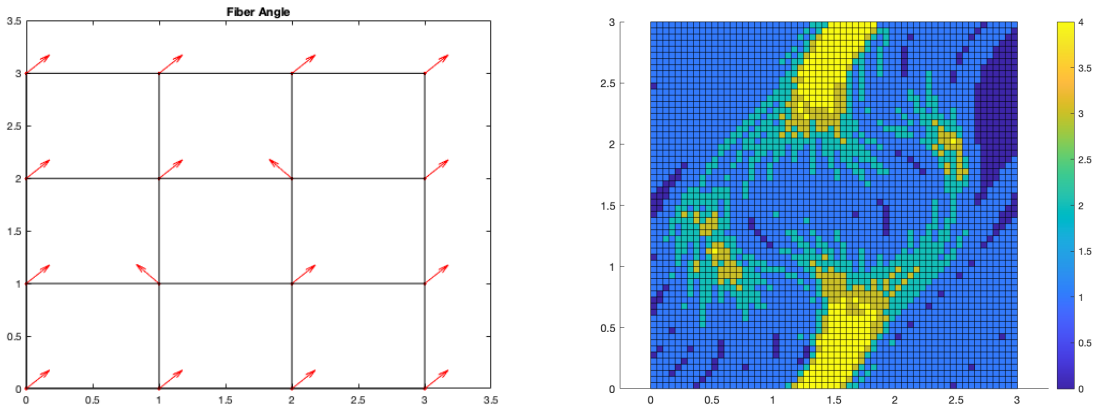


Figure 6.6. Test case where 2 center diagonal angles are rotated from 0° to 180° while all other angles are held at 45°.

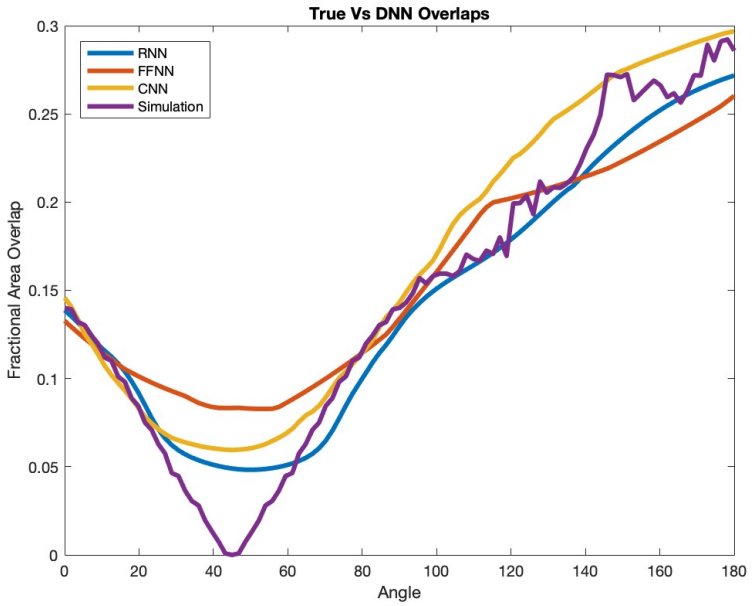


Figure 6.7. A functional test of the performance of the feed-forward, convolutional, and recurrent neural networks for the test case presented in Figure 6.6

tested with 3 datasets generated using 3 independent Latin Hyper-Cubed distributions with sample sizes of 500,000, 60,0000, and 10,000 size, respectively. The predicted accuracy of the fitted models at the test sample points (10,000) are quantified by the Mean Absolute Percent Error and the Coefficient of Determination, R^2. We show our results in Table 6.2.

The second problem domain of the plate with a hole in the center is initially not enhanced with divergence and curl. Calculating the divergence for non-uniform grids is expensive when compared to a rectangle grid. Taking on a average .04 seconds to compute with current libraries. A comparison is presented at the end of the section to compare the performance of the models with and without the curl and divergence data.

Table 6.2. Neural Network Performance Results for Problem 2

	'Known True' Average Wall Clock Time Per Sample	Mean Absolute Percentage Error	R^2	'ML Prediction' Average Wall Clock Time Per Sample
Feed Forward Network	.225 Secs	24.333%	0.33848273	6.489e-5 Secs
Recurrent Network	.225 Secs	24.08%	0.35066851	5.89e-5 Secs

The final models have the architecture and hyper parameters detailed in Table 5.3.2.

From Table **??** we can see that the recurrent neural network performed only slightly better on the test dataset in terms of R^2. Both models achieved similar performance in terms of the Mean Average Error Percentage. The speed up achieved is comparable to the first problem of the flat plate.

Figures 6.8 and 6.9 are scatter plots of true versus predicted values of the overlaps on the surface. The plotted data in the scatter plots for the RNN and FFNN for Problem 2 are noticably not plotted along the 45 degree line, nor as tightly grouped together as the results for Problem 1, presented in the scatter plots in Figures 6.1, 6.2, and 6.3. This visually confirms the drop in performance also communicated by our test metrics.

6.1.2.1 Test 1

The neural networks for Problem 2 do not perform as well as for Problem 1. To investigate we construct 2 test cases in a similar fashion as in Problem 1 to perform a qualitative analysis and determine the usefulness of the neural networks.

In Figure 6.10 we present a constructed example where we hold all angles fixed at 0° from the horizontal and the two angles inside the left top and bottom corners are rotated -90° to 90°. In Figure 6.10 these two angles are illustrated at -90°.

We expect to see zero, or close to zero, fractional overlap area when the test angles match the fixed angles. That is when all angles are aligned equal at 0°. As the test angles move towards and away from this alignment, we would expect that the induced errors would subside and increase appropriately.

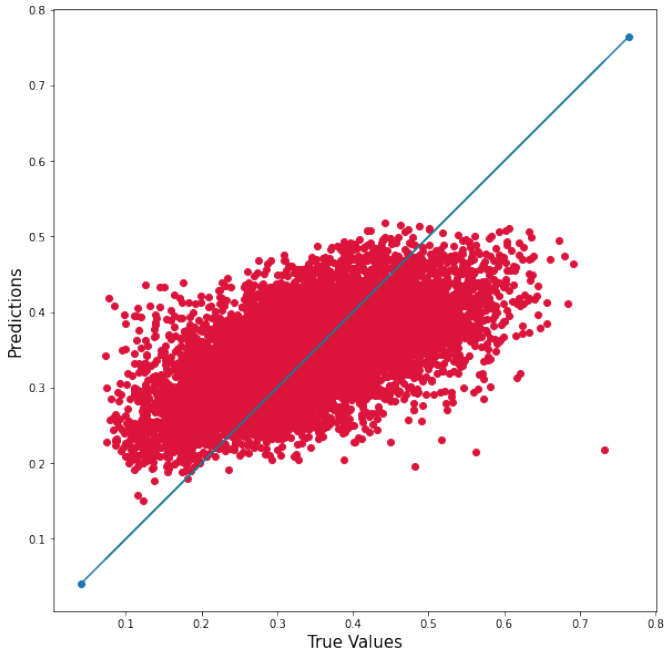


Figure 6.8. A scatter plot of predicted versus true value for feed forward neural network on the test dataset for Problem 2.

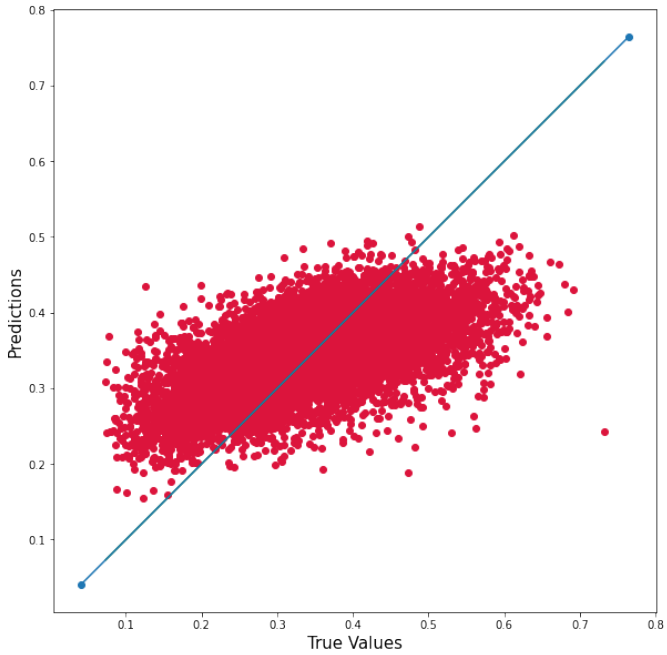


Figure 6.9. A scatter plot of predicted versus true value for recurrent neural network on the test dataset for Problem 2.

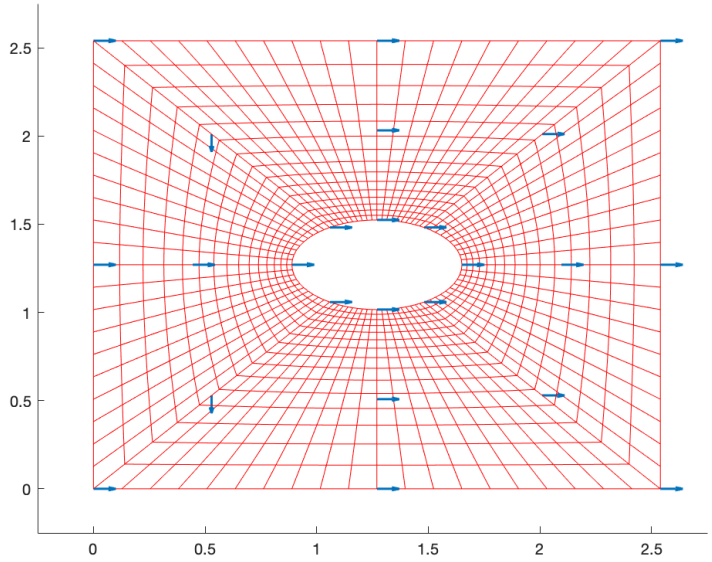


Figure 6.10. Top and bottom angles on the left inside corners are rotated -90° to 90°.

The graph in Figure 6.11 qualitatively portrays better performance by the neural network models than is communicated by the raw metrics in Table **??**. The functional output of the original simulation algorithm is also highly non-linear when compared to the original problem of a flat plate. The non-linear behavior of Problem 2 would require a model with more complexity or neurons than that needed to represent Problem 1. The neural networks capture the general trend of the underlying function. Similar to the argument presented in Problem 1, the model does not make predictions for small overlap values. The positive bias of the predictions may be attributed to the limited number of low overlap samples present in the training datasets. We also notice that in addition to the noisy behavior of our underlying function the function is not zero when the control angles are oriented at 0°. This indicates that the underlying simulation function may be failing to find the optimum solution and this inconsistency may also account for the poorer performance of the DNNS.

In Figure 6.12 we compare the performance of two RNNs with the same hyper parameters as detailed in Table 5.3.2 but one model is trained on a dataset of ply angles and the other model is trained on that same dataset but enhanced with the local curl and divergence data. While the models do not make the same predictions no strong conclusion can be drawn.

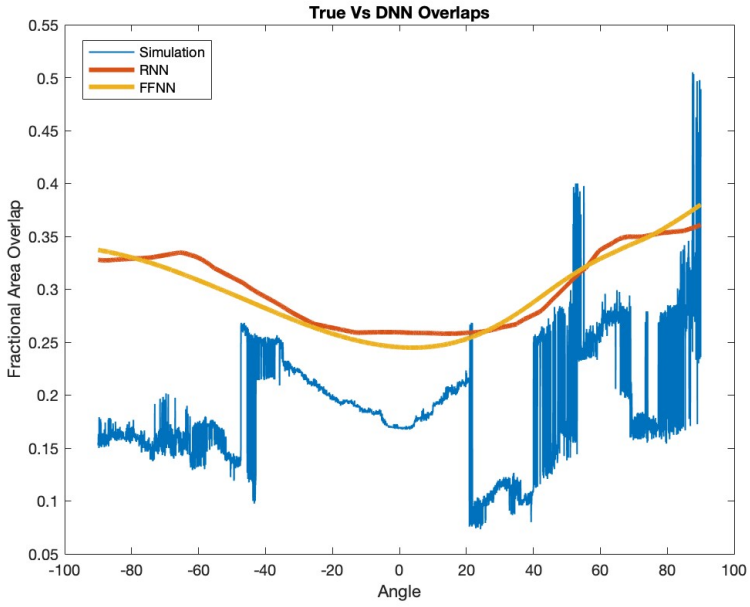


Figure 6.11. A comparison of results between the feed-forward neural network, the recurrent neural network, and the simulation algorithm for the test case presented in Figure 6.10

6.1.2.2 Test 2

In Figure 6.13 we present a constructed example where we hold all angles fixed at 0° from the horizontal and the two angles inside the right top and bottom corners are counter rotated from -90° to 90° and vice versa. The top angle is rotated clockwise and the bottom angle is rotated counter clockwise. In Figure 6.13 these two angles are illustrated at their initial orientation -90° for the bottom angle and 90° for the top angle.

The models qualitatively perform similar in this second test as presented in Figures 6.14. The model does not seem to capture the behavior of the first 60° of arc. The positive bias is also greater in this test than in the first test case. Interestingly, the input data seems to be less noisy than in Test 1. This may be due to the fact that the overall tow direction is from left to right and the test angles are on the right side of the hole. While the underlying function may be less noisy this does not translate into better performance of the models.

As was conducted for test 1 in Figure 6.15 we compare the performance of two RNNs with the same hyper parameters as detailed in Table 5.3.2 but one model is trained on a dataset of ply angles and the other model is trained on that same dataset but enhanced with the local curl and divergence data. Again both models perform

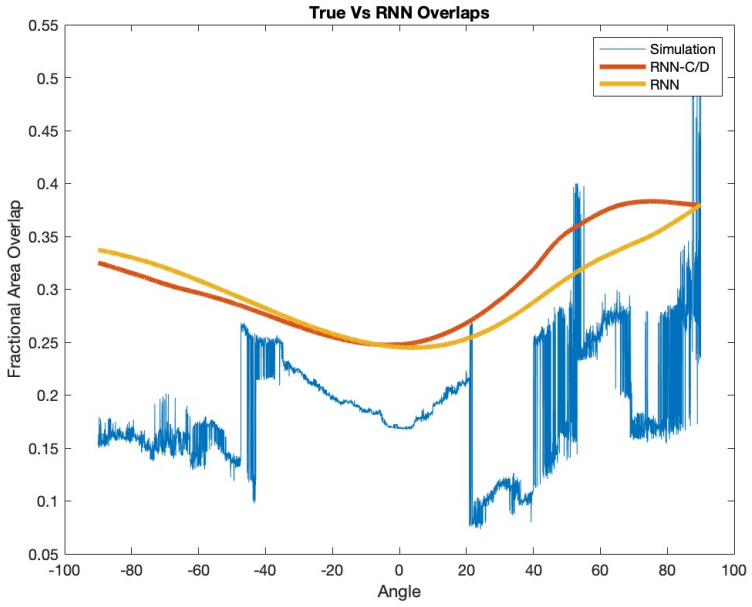


Figure 6.12. A comparison of results between the recurrent neural network trained on data with inputs of coarse fiber angles and data with inputs of coarse fiber angles, curl, and divergence for the test case presented in Figure 6.10

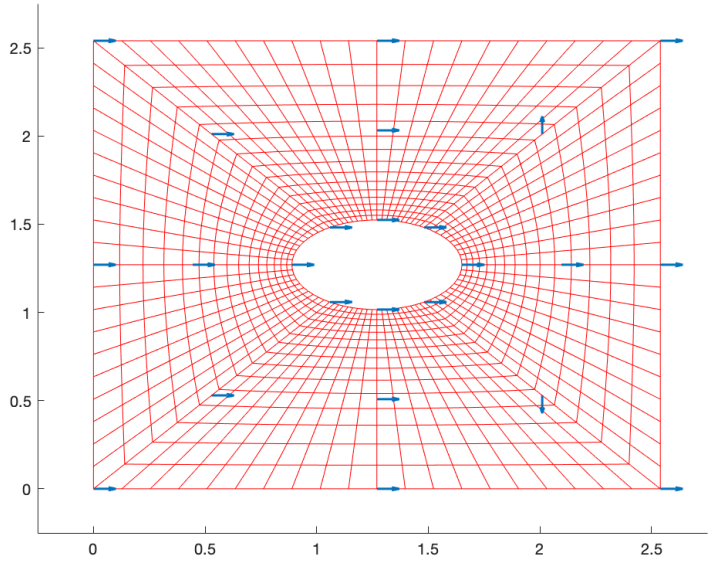


Figure 6.13. The two angles inside the right top and bottom corners are counter rotated from -90° to 90°

similiarly enough that no strong conclusion can be made. Since the cost of computing the curl and divergence for a sample is .04 degrees and the difference in performance is not appreciably different we did not include the curl and divergence in other model comparisons.

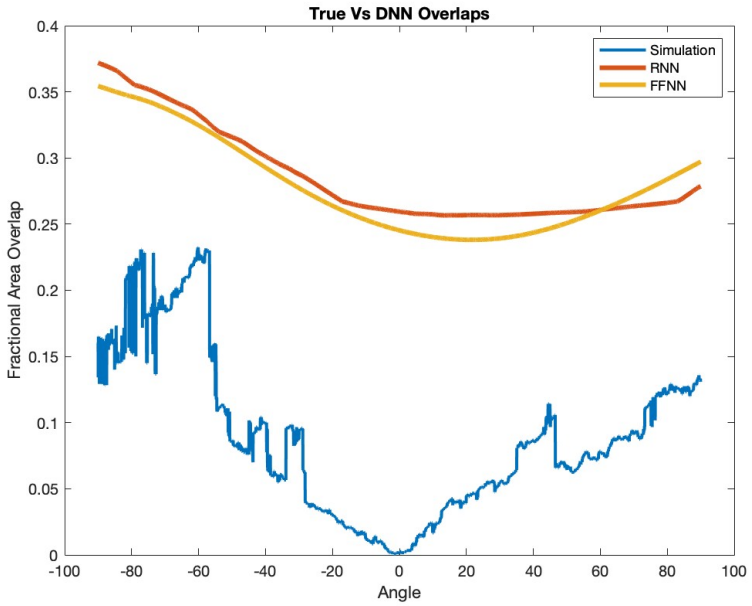


Figure 6.14. A comparison of results between the feed forward neural network, the recurrent neural network, and the simulation algorithm for the test case presented in Figure 6.13

6.1.3 Box-Behnken Designs

A possible explanation for the positive bias seen when comparing the model predictions to the simulation/path planning algorithm is that the LHS designs, in seeking to evenly sample the design space, do not generate data samples where the majority of ply angles have the same value. But in both the constructed test cases most ply angles are oriented in the same direction.

In order to create samples with this characteristic we explore other DOE methods. Central Composite design is a good candidate but suffers from the curse of dimensionality. The number of samples for a full CCD is $2^k + 2k + N_0$ where N_0 is a constant. For $K = 24$, the number of factors in our second problem, this is approximately 16 million samples. Due to the computational expense of creating data samples this is a prohibitively high number.

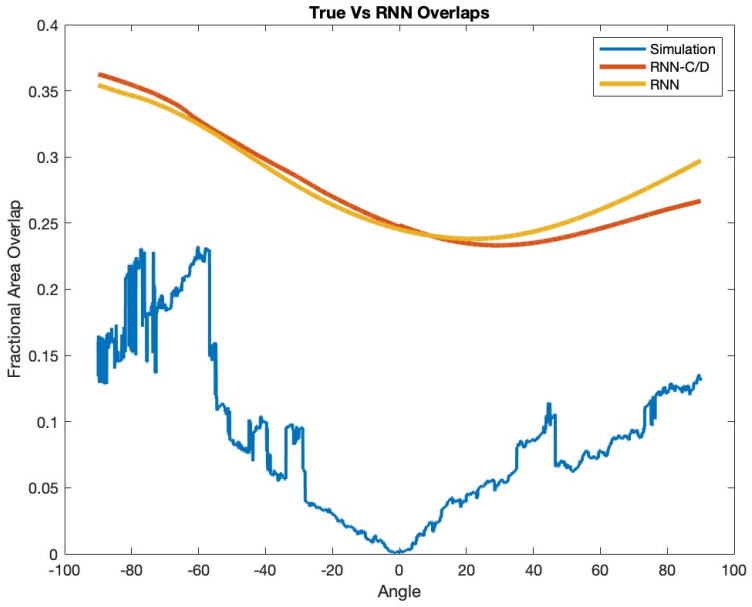


Figure 6.15. A comparison of results between the recurrent neural network trained on data with inputs of coarse fiber angles and data with inputs of coarse fiber angles, curl, and divergence for the test case presented in Figure 6.13

The Box-Behnken Design is another good candidate. It does not suffer from the curse of dimensionality like CCD. The number of samples for a full BBD is $2k(k-1)+N_0$. For $K=24$ this is 1105 sample points. Here we have the opposite problem from the CCD in that we have too few samples to add to the dataset to have a large impact. The Box-Behnken Design samples the midpoints of the edges of the design space. We create an expanded BBD explained below.

In Figure 6.16 we see a visualization of a BBD for 3 factors. These same sample points are listed in Table 6.3. For the points in Figure 6.16 α is equal to 0. This is the traditional BBD where only the midpoints of the edges are sampled. We expand the Box-Behnken design by sampling the entirety of the edges of the design space. We do this by tracing α from $[-1,1]$.

Using this method we can conveniently choose the resolution we use to create these expanded datasets. In Problem 1 the ply angles vary from 0° to 180° and in Problem 2 the ply angles vary from -90° to 90°. We create BBDs every 2 degrees or $\alpha=[0°:2°:180°]$ for Problem 1 and $\alpha=[-90°:2°:90°]$ for Problem 2. This allows us to create 43200 samples for Problem 1 and 99360 samples for Problem 2. We combined these data sets with the training data sets of 500,000 samples. We then take

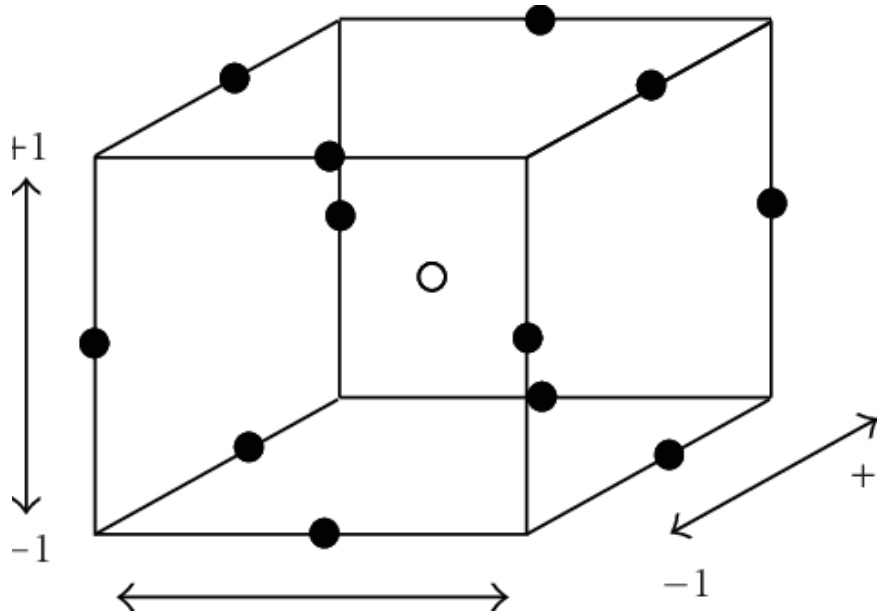


Figure 6.16. A Box-Behnken Design presented visually in 3 dimensions. The BBD samples the midpoints of the edges of the design space. [K. Kandananond, Using the response surface method to optimize the turning process of aisi 12l14 steel, Advances in Mechanical Engineering, 2 (2010), p. 362406.]

Table 6.3. Expanded Box-Behnken Design for 3 Factors and $N_0 = 0$

Experiment	X_1	X_2	X_3
1	-1	-1	α
2	1	-1	α
2	1	-1	α
3	-1	1	α
4	1	1	α
5	-1	α	-1
6	-1	α	1
7	1	α	-1
8	1	α	1
9	α	-1	-1
10	α	1	-1
11	α	-1	1
12	α	1	1

our trained models and train them for an additional 50 epochs on the composite dataset. For Problem 1 we show results also for a composite dataset of a traditional BBD and our LHS training dataset.

From our results shown in Figures 6.17, 6.18, 6.19, 6.20, 6.21, 6.22, 6.23, and 6.24 we can see that the expanded BBD samples broadly helped our models lower the bias of their predictions with out losing the ability to predict the general behavior of the simulation. The notable exception is for Problem 2, Test 2, the RNN model performs worse.

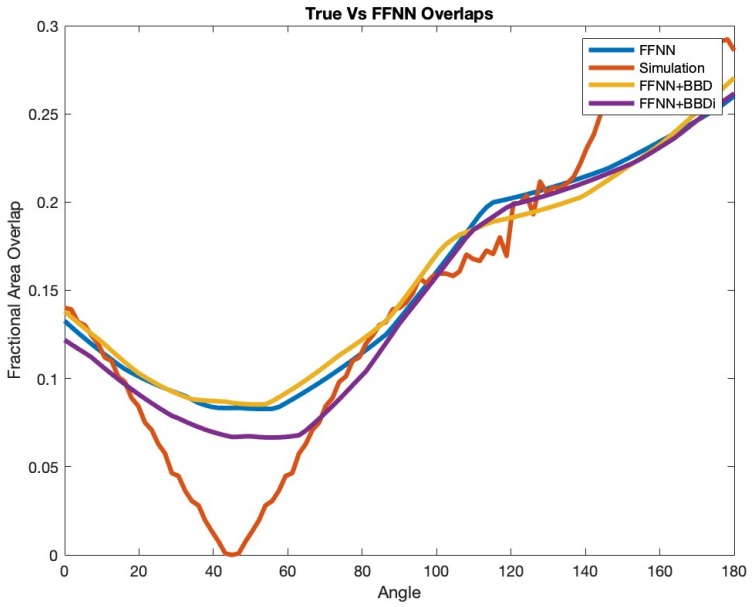


Figure 6.17. A comparison of results between the FFNN rained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.4

6.1.4 Composite Data

In a last exploration in order to improve our results for Problem 2 we combine all the generated data in one large dataset and train our model on this dataset. We use K-fold Cross Validation to determine our MAPE and R^2 values. In Table 6.4 we can see that our results are not grealty changed. By examining Figures 6.25 and 6.26 we can see an improvement over just adding the BBD data alone to the original training dataset.

6.2 Discussion

The general results shared in Tables 6.1 and 6.2 and the specific performance displayed in Tests 1 and 2 for both problem domains provide the basis for an argument of the feasibility of using Machine Learning to surrogate the tow path creation

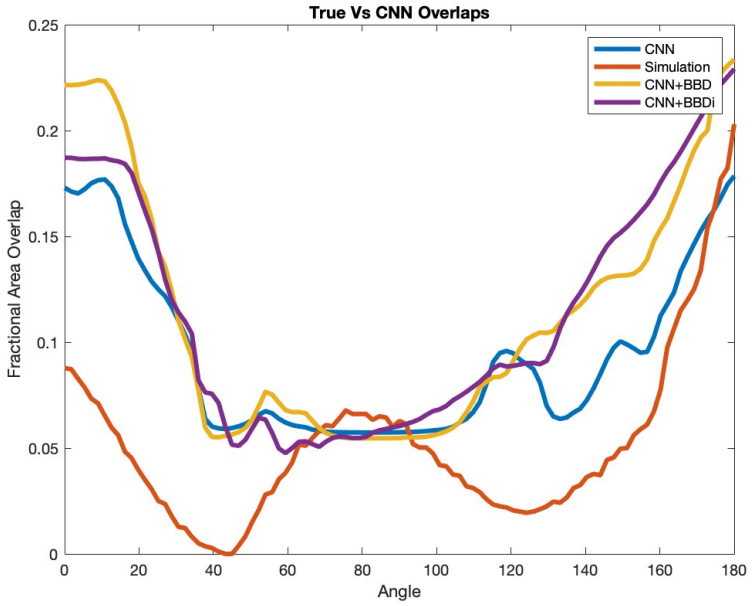


Figure 6.18. A comparison of results between the CNN rained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.4

Table 6.4. Neural Network Performance Results for Problem 2 On Combined Dataset

	'Known True' Average Wall Clock Time Per Sample	Mean Absolute Percentage Error	R^2	'ML Prediction' Average Wall Clock Time Per Sample
Feed Forward Network	.225 Secs	25.01%	0.339	6.489e-5 Secs
Recurrent Network	.225 Secs	23.66%	0.368	5.89e-5 Secs

simulation algorithms in an overall optimization scheme. These results provide enough evidence to support refinement of the current models and further research into applying Machine Learning techniques to this problem domain.

For problem 1 in Figures 6.5 we can see that while the models do not perfectly trace the underlying function, they are representing the general behavior. If the goal of the simulation algorithm is to inform the overall optimization whether the current set of test angles are better or worse than the last set of angles, than the surrogate models can provide a reasonable estimation of that change. Specifically for the recurrent neural network the troughs, peaks and slopes are most closely aligned.

In the more complicated problem 2 in Figures 6.11 and 6.14 we can see the underlying function is more noisy. The decreased performance of the neural networks shows that with increased complexity in the problem domain, the neural networks must

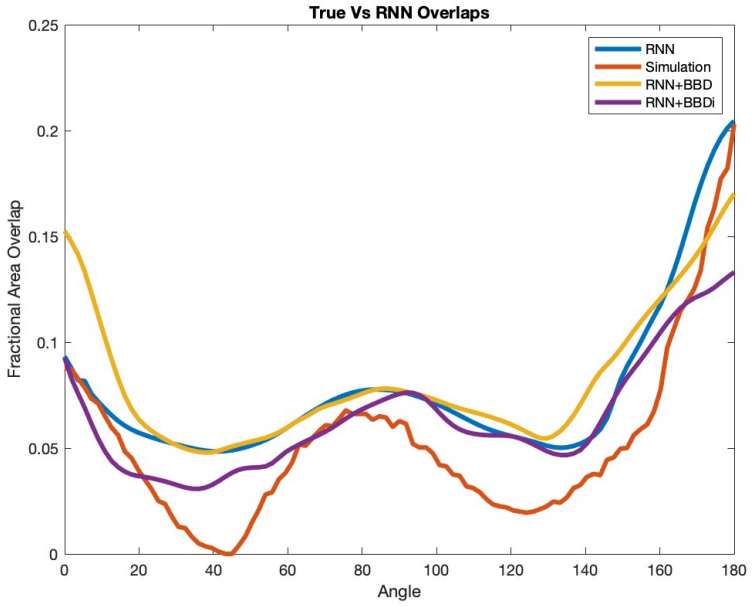


Figure 6.19. A comparison of results between the RNN rained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.4

become more complex, and thus expensive, in order to capture this behavior. This is a drawback for using this technique. If the time to develop, perform a hyper parameter search, train and validate a neural network model exceed the time saved in the overall optimization, then it is not a useful technique.

The expanded Box-Behnken design method allowed us to sample the edges of the design space. This helped the models to improve their performance in the regions of low overlaps. When we combined all the data together and trained the model we can see that the additional LHS design samples reduced the effectiveness of the BBD samples to lower prediction bias. But these composite models visually seem to predict the minimums of the test cases better than the original 500,000 sample training data combined with the BBD samples.

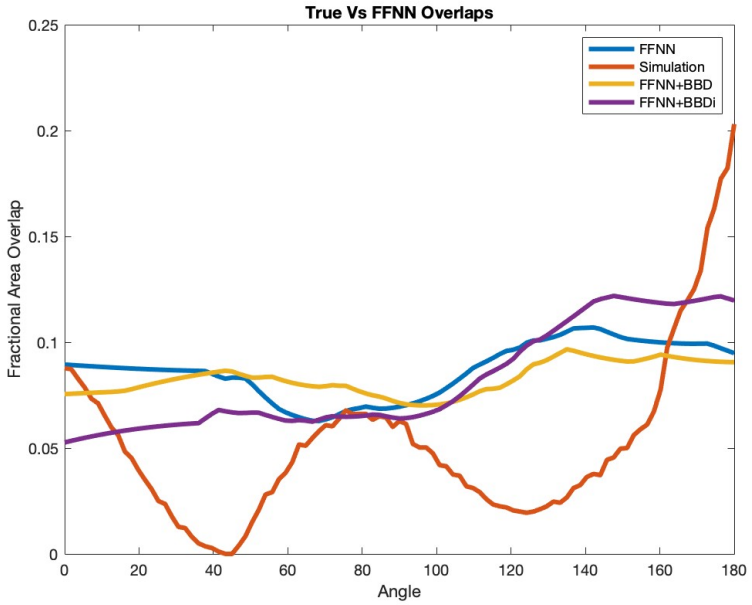


Figure 6.20. A comparison of results between the FFNN rained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.6

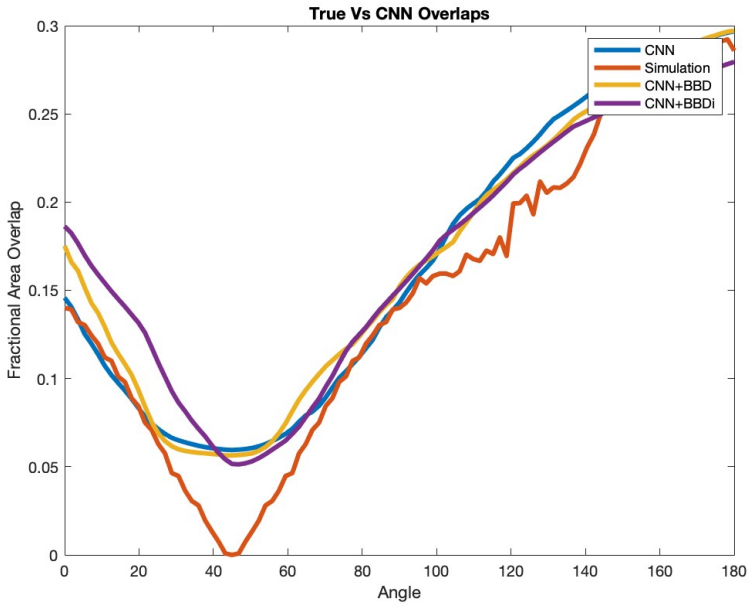


Figure 6.21. A comparison of results between the CNN rained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.6

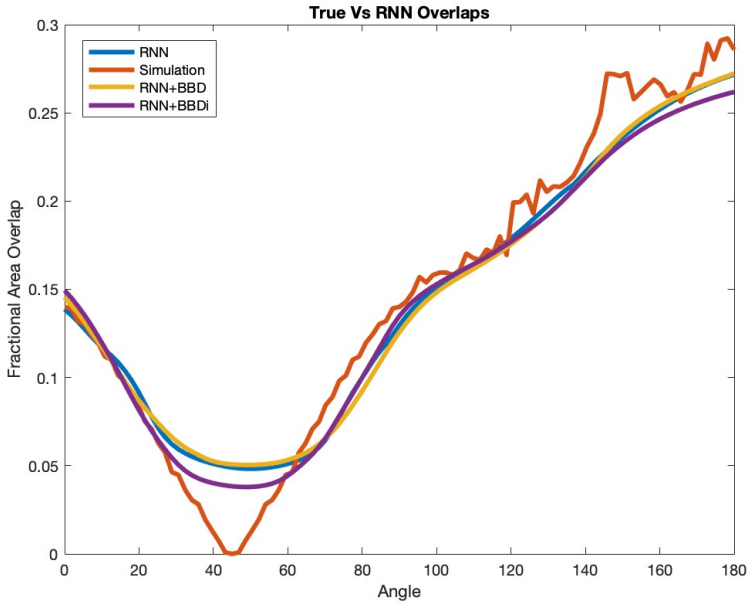


Figure 6.22. A comparison of results between the RNN rained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.6

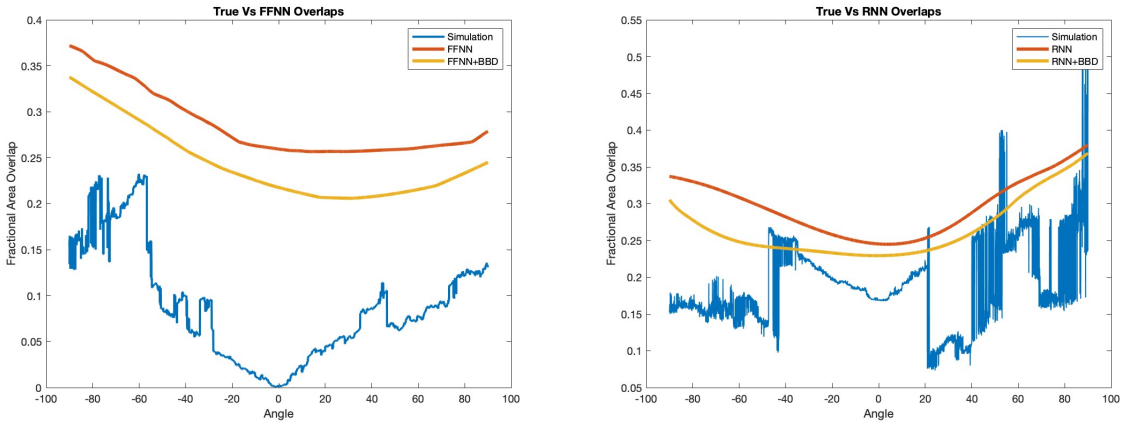


Figure 6.23. A comparison of results between the FFNN and RNN rained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.10

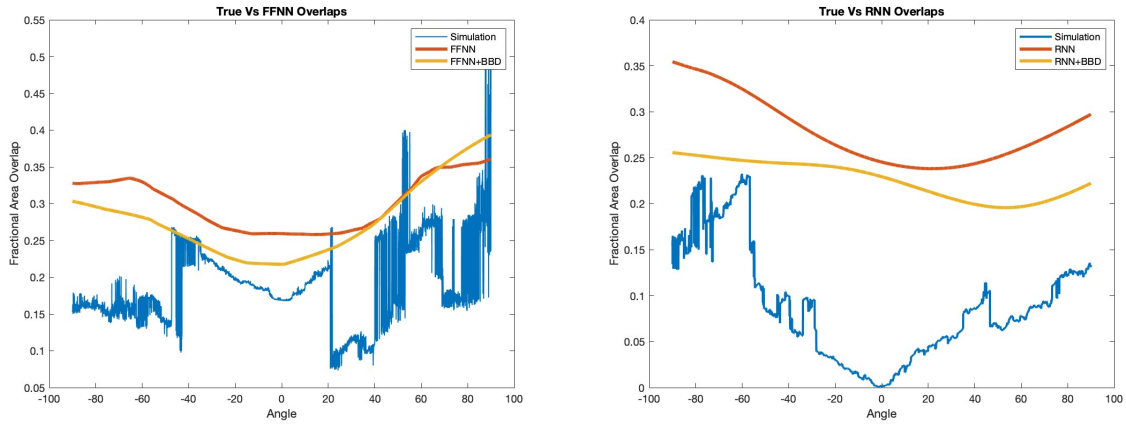


Figure 6.24. A comparison of results between the FFNN and RNN trained on additional samples from a BBD, expanded BBD, and the simulation algorithm for the test case presented in Figure 6.13

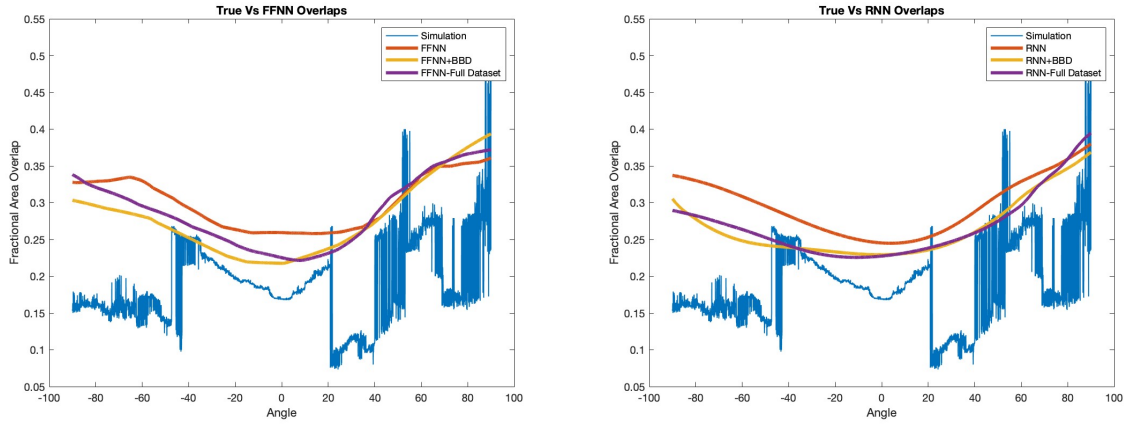


Figure 6.25. A comparison of results between the FFNN and RNN trained on a composite dataset and the simulation path planning algorithm for the test case presented in Figure 6.10

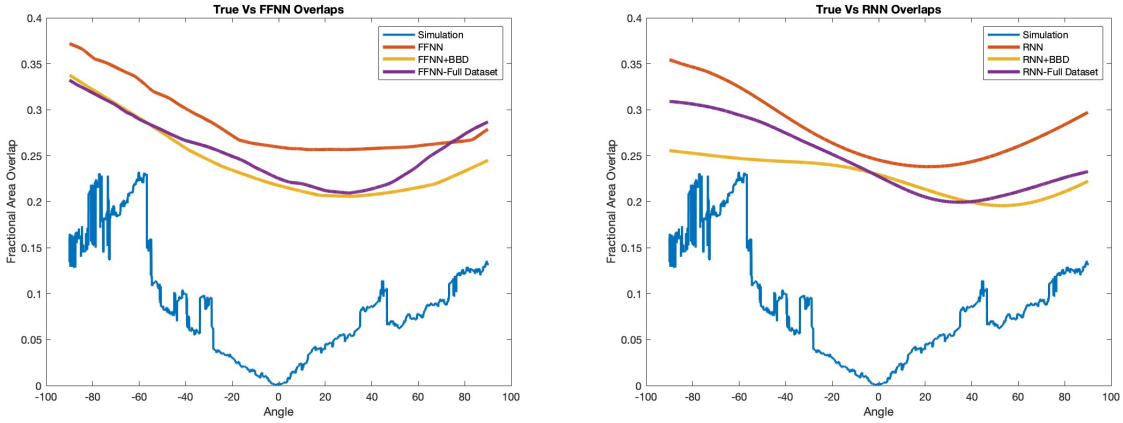


Figure 6.26. A comparison of results between the FFNN and RNN trained on a composite dataset and the simulation path planning algorithm for the test case presented in Figure 6.13

CHAPTER 7
CONCLUSIONS AND RECOMMENDATIONS

This Thesis has shown the viability of using deep neural networks to surrogate a program that simulates the manufacturing of a composite structure by an Automated Fiber Placement machine. This research explored the drawbacks and potential benefits of both the process of creating neural networks and implementing them as surrogate models.

7.1 Conclusions

Because this application is not a standard neural network problem, choosing the right DNN architecture required understanding of both the problem domain and the functional abilities of neural networks. Initially, we looked at the coarse fiber angles as a static structure, but by conceptually re-framing the problem as sequential data, we experimented with and were able to achieve the best results using a recurrent neural network.

The process of performing the functional analysis and comparison of the neural networks with the simulation algorithm allowed us to better understand what the DNNs were learning. Areas where the DNNs struggled were identified as areas to improve the underlying simulation algorithm. This artifact of studying the performance of the DNNs demonstrates that the process of creating a surrogate is useful for understanding a problem.

The DNNs did not perfectly surrogate the simulation algorithm. The results were good enough to show that the method is viable as a surrogate model. The functional analysis plots shown in this chapter show the DNNs capture the functional change in behavior. The overall goal of this Thesis is to provide a constraint to a larger optimization problem that at a high level answers the question, when a change is made to the fiber angles, is the structure more or less manufacturable? For Problem 1 the RNN produces a result approximately 2900 times faster with a MAPE of 6.225% . This loss of information may be acceptable with such a considerable speed up in prediction time.

For Problem 2 both the FFNN and RNN MAPE drops to 24%. This thesis showed that as the complexity of the structure increased the neural network performance decreased.

Deep neural networks are most popular in computer vision, natural language processing, and user attention applications, but in recent years the wealth of resources, tools, and libraries have allowed researchers to experiment with applying Machine Learning to other domains of science. Neural networks have been used to detect defects in real time by an AFP machine [9]. But utilizing ML techniques to aid in the design optimization process has not been explored before. This thesis has demonstrated that the use of DNNs to successfully surrogate path planning simulations is feasible. As this research attempted no more than a proof of concept, there is further research to be done. The difference in performance between the Problem 1 and Problem 2 demonstrate the difficulties in generalizing the results of DNNs. Any implementation in a real world production environment would require adjustment and fine tuning of the neural network models.

7.2 Recommendations

Further research should focus on exploring new ML architectures, improving data generation, and the mechanism/techniques to incorporate DNN surrogates into composite optimization loops. Our results showed the best performance with RNNs and further research into the use of deeper LSTMs and bidirectional LSTMs would be worth investigating. Also Gated Recurrent Units are a computationally cheaper alternative to LSTMs. They may allow for a deeper RNN without increasing the training cost.

Transformers are the modern state-of-the-art DNN to answer sequential problems, but they are not easy to implement and would be a larger research project. The input data for this problem is naturally represented as a graph of nodes and edges. Graph neural networks are becoming more important in the field of ML and would be a good candidate for further research.

The implementation constraints of convolutional neural networks make them a poor choice from an software engineering perspective. Also, CNNs were made for detecting and classifying objects in images. This conclusion by Yann LeCun: "The local statistics of images and other signals are invariant to location. In other words, if a motif can appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array [23]" highlights a major conceptual challenge in further application of CNNs to this problem. Learning features that are invariant to location

may be a negative aspect to this method since the relationships of the ply angles are not invariant to position.

In Chapter 6, we see that all of the models struggle with making accurate predictions of surfaces with a low amount of overlaps. Creating better datasets that contain samples of these lower overlap inputs would help train the models to make better predictions in these regions. We attempted to remedy this by creating addition samples with an expanded BBD scheme. While this resulted in moderate success, it does affirm the benefit of creating more samples that occupy regions of poor performance.

An adaptive sampling technique as detailed in [13] could help in training the model with less data to make more accurate predictions in these regions. In order to implement this, the workflow for the surrogate model and the DNNs would need to be unified. The computation cost of transferring large amounts of data between MATLAB is prohibitive to this method.

We can try to improve the sampling methods by improving the LHS method as proposed in [6, 38], or by using a centroidal voronoi tessellation space filling technique as detailed in [35, 41]. We can also combine the LHS design with a central composite cesign in order to provide the dataset with more edge cases. This would allow us to use the data already generated but simply add more samples.

Finally the methods that are used to incorporate the surrogate models into the overall structural optimization need to be refined. Since complex structures decrease the performance of the DNN models it may be sensible to divide the surfaces into less complex sections. This would require fewer DNNs to be trained, but how the surfaces would be divided and what the DNNs would take as input and generate as output would be non-trivial decisions.

BIBLIOGRAPHY

[1] *Pytorch documentation torch.nn.*

[2] *Ibm cloud education: What are convolutional neural networks?*, Oct 2020.

[3] *Latin hypercube sampling (lhs)*, Latin Hypercube Sampling (LHS) - EVOCD, (2021).

[4] M. Abramson, C. Audet, and J. Dennis, *Optimization using surrogates for engineering design c*, (2003).

[5] M. Albazzan, *Efficient Design Optimization Methodology for Manufacturable Variable Stiffness Laminated Composite Structures*, PhD dissertation, University of South Carolina, Mechanical Engineering, 2020.

[6] B. Beachkofski and R. Grandhi, *Improved distributed hypercube sampling*, American Inst of Aeronautics and Astronautics, AIAA, (2002).

[7] G. Bontempi, *"Statistical Foundations of Machine Learning: The Handbook"*, 02 2021.

[8] T. R. Brooks and J. R. Martins, *On manufacturing constraints for tow-steered composite design optimization*, Composite Structures, 204 (2018), pp. 548–559.

[9] J. Brüning, B. Denkena, M.-A. Dittrich, and T. Hocke, *Machine learning approach for optimization of automated fiber placement processes*, Procedia CIRP, 66 (2017), pp. 74–78. 1st CIRP Conference on Composite Materials Parts Manufacturing (CIRP CCMPM 2017).

[10] D. Chicco, M. J. Warrens, and G. Jurman, *The coefficient of determination r-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation*, PeerJ Comput Sci, 7 (2021), p. e623.

[11] A. de Myttenaere, B. Golden, B. Le Grand, and F. Rossi, *Mean absolute percentage error for regression models*, Neurocomputing, 192 (2016), pp. 38–48. Advances in artificial neural networks, machine learning and computational intelligence.

[12] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, *Incorporating second-order functional knowledge for better option pricing*, in Advances in Neural Information Processing Systems, T. Leen, T. Dietterich, and V. Tresp, eds., vol. 13, MIT Press, 2000.

[13] J. Eason and S. Cremaschi, *Adaptive sequential sampling for surrogate model generation with artificial neural networks*, Computers Chemical Engineering, 68 (2014), pp. 220–232.

[14] K. Fukushima, *Cognitron: A self-organizing multilayered neural network*, Biological Cybernetics, 20 (1975), pp. 121–136.

[15] F. Gers and E. Schmidhuber, *Lstm recurrent networks learn simple context-free and context-sensitive languages*, IEEE Transactions on Neural Networks, 12 (2001), pp. 1333–1340.

[16] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in AISTATS, 2011.

[17] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*, O'Reilly Media, Sebastopol, CA, 2017.

[18] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Comput., 9 (1997), p. 1735–1780.

[19] R. Jozefowicz, W. Zaremba, and I. Sutskever, *An empirical exploration of recurrent network architectures*, in Proceedings of the 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings of Machine Learning Research, Lille, France, 07–09 Jul 2015, PMLR, pp. 2342–2350.

[20] K. Kandananond, *Using the response surface method to optimize the turning process of aisi 12l14 steel*, Advances in Mechanical Engineering, 2 (2010), p. 362406.

[21] S. M. Karazi, M. Moradi, and K. Y. Benyounis, *Statistical and numerical approaches for modeling and optimizing laser micromachining process-review*, in Reference Module in Materials Science and Materials Engineering, Elsevier, 2019.

[22] D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, International Conference on Learning Representations, (2014).

[23] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[24] L. Liang, M. Liu, C. Martin, and W. Sun, *A deep learning approach to estimate stress distribution: A fast and accurate surrogate of finite-element analysis*, Journal of The Royal Society Interface, 15 (2018).

[25] J. Marrouzé, J. M. Housner, and F. Abdi, *Effect of manufacturing defects and their uncertainties on strength and stability of stiffened panels*, Montreal, CA, USA, 28 July 2013, THE 19TH INTERNATIONAL CONFERENCE ON COMPOSITE MATERIALS.

[26] M. D. McKay, R. J. Beckman, and W. J. Conover, *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code*, Technometrics, 21 (1979), pp. 239–245. Full publication date: May, 1979.

[27] S. K. R. A. M. H. D. Minaya C., Venkataraman, *Tow path recovery and*

quantification of manufacturing constraints for tow steered composite structures, San Diego, CA, June 2023, AIAA Aviation Forum.

[28] O. A. Montesinos López, A. Montesinos López, and J. Crossa, *Fundamentals of Artificial Neural Networks and Deep Learning*, Springer International Publishing, Cham, 2022, pp. 379–425.

[29] V. Nair and G. Hinton, *Rectified linear units improve restricted boltzmann machines vinod nair*, vol. 27, 06 2010, pp. 807–814.

[30] Y. Nawib, *Classification and applications of textile composite materials: Part 1*, Dec 2009.

[31] A. Olsson, G. Sandberg, and O. Dahlblom, *On latin hypercube sampling for structural reliability analysis*, Structural Safety, 25 (2003), pp. 47–68.

[32] J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C.-H. Lee, *On mean absolute error for deep neural network based vector-to-vector regression*, IEEE Signal Processing Letters, 27 (2020), pp. 1485–1489.

[33] P. Ramu, P. Thananjayan, E. Acar, G. Bayrak, J. W. Park, and I. Lee, *A survey of machine learning techniques in structural and multidisciplinary optimization*, Structural and Multidisciplinary Optimization, 65 (2022), p. 266.

[34] K. S. Ravi Chandran, *Review: Fatigue of fiber-reinforced composites, damage and failure*, Journal of the Indian Institute of Science, 102 (2022), pp. 439–460.

[35] V. Romero, J. Burkardt, M. Gunzburger, J. Peterson, and T. Krishnamurthy, *Initial application and evaluation of a promising new sampling method for response surface generation: Centroidal voronoi tessellation*, 04 2003.

[36] V. Romero, L. Swiler, and A. Giunta, *Construction of response surfaces based on progressive-lattice-sampling experimental design with application to uncertainty propagation*, Structural Safety, 26 (2004), pp. 201–219.

[37] A. Sagheer and M. Kotb, *Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems*, Scientific Reports, 9 (2019), p. 19038.

[38] Y. Saka, M. GUNZBURGER, and J. Burkardt, *Latinized, improved lhs, and cvt point sets in hypercubes*, IEEE Transactions on Information Theory - TIT, 4 (2007).

[39] T. Simpson and D. Lin, *Sampling strategies for computer experiments: Design and analysis*, Int. J. Reliab. Appl., 2 (2001).

[40] S. Sun, Z. Han, H. Fu, H. Jin, J. S. Dhupia, and Y. Wang, *Defect characteristics and online detection techniques during manufacturing of frps using automated fiber placement: A review*, Polymers, 12 (2020).

[41] L. Swiler, R. Slepoy, and A. Giunta, *Evaluation of Sampling Methods in Constructing Response Surface Approximations*.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2017.

[43] F. A. C. Viana, C. Gogu, and T. Goel, *Surrogate modeling: tricks that endured the test of time and some recent developments*, Structural and Multidisciplinary Optimization, 64 (2021), pp. 2881–2908.

[44] Y. Wang, D. Oyen, W. G. Guo, A. Mehta, C. B. Scott, N. Panda, M. G. Fernández-Godino, G. Srinivasan, and X. Yue, *Stressnet - deep learning to predict stress with fracture propagation in brittle materials*, npj Materials Degradation, 5 (2021), p. 6.

[45] S. Wright, *Correlation and causation*, (1921), pp. p. 557–585.–USDA.

[46] K. Wu, B. Tatting, B. Smith, R. Stevens, G. Occhipinti, J. Swift, D. Achary, and R. Thornburgh, *Design and manufacturing of tow-steered composite shells using fiber placement*, 05 2009.

[47] L. Zhang, X. Wang, J. Pei, and Y. Zhou, *Review of automated fibre placement and its prospects for advanced composites*, Journal of Materials Science, 55 (2020), pp. 7121–7155.

[48] R. Ñanculef, P. Radeva, and S. Balocco, *Chapter 9 - training convolutional nets to detect calcified plaque in ivus sequences*, in Intravascular Ultrasound, S. Balocco, ed., Elsevier, 2020, pp. 141–158.