UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title
Machine Learning-based Inverse Solution for Predictions of Impact Conditions during Car Collisions

Permalink
https://escholarship.org/uc/item/77j0k7nb

Author
Li, Tiange

Publication Date
2019

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Machine Learning-based Inverse Solution for Predictions of Impact Conditions during Car Collisions

by

Tiange Li

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Civil and Environmental Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Shaofan Li, Chair
Professor Khalid M. Mosalam
Professor Lin Lin

Spring 2019

Machine Learning-based Inverse Solution for Predictions of Impact Conditions during Car Collisions

Copyright 2019
by
Tiange Li

Abstract

Machine Learning-based Inverse Solution for Predictions of Impact Conditions during Car Collisions

by

Tiange Li

Doctor of Philosophy in Civil and Environmental Engineering

University of California, Berkeley

Professor Shaofan Li, Chair

In this work, a novel deep learning computational framework is developed to determine and identify the damage load conditions of different types of structures, including cantilever beams of inelastic materials, elasto-plastic shell structures of inelastic materials and crashed cars subjected to mechanical forcing actions.

There are a variety of methods to measure engineering responses based on the corresponding load conditions. The aim of this work is to establish reverse analysis algorithms. This artificial intelligence framework offers a practical solution to solve the inverse problem of engineering failure analysis based on final material and structure damage states and fields. More precisely, the machine learning inverse problem solver may be a practical solution to characterize failure load parameters and conditions based on the permanent plastic deformation distribution or the residual displacement condition of beam, shell structures and cars.

The study presents the detailed machine learning algorithm, data acquisition and learning processes, and validation and verification examples. Neural Network Modeling offers a cohesive approach to the computational mechanical problems based on TensorFlow. Different activation functions and loss functions are compared theoretically and numerically during implementing neural network. Feature selection is used in model construction for simplification of models to make them easier to interpret and lead to shorter training times. It is demonstrated that the developed machine learning algorithm can accurately identify a practically unique prior static loading as well as impact loading state for different structures, in an inverse manner, using the permanent plastic deformation or the residual displacement as the forensic signatures.

The data-driven based method developed in this work, employs Artificial Neural Networks to provide a powerful tool for forensically diagnosing, determining, and identifying damage loading conditions for engineering structures in accidental failure events, such as car crashes and infrastructure or building structure collapses. The machine learning inverse problem solver developed

here may have potential broader impacts on general forensic material and structure analysis using permanent plastic deformations.

I dedicate this dissertation to my family, whose love and guidance made me into the person I am today.

Contents

Contents **ii**

List of Figures **iv**

List of Tables **vi**

1 Introduction **1**
1.1 Background 1
1.2 Thesis Organization 3

2 Related Work and Problem Statement **5**
2.1 Summary of Related Machine Learning Work 5
2.2 Inverse Problem in Computational Mechanics 6
2.3 Problem statement 7

3 Machine Learning Methodology **9**
3.1 General Machine Learning Methodology 9
3.2 Methodology for Cantilever Beams with Inelastic Materials 19
3.3 Methodology for Elasto-plastic Shell Structures 26
3.4 Methodology for Car Collisions 32

4 Theoretical Contribution to Algorithms **36**
4.1 Activation Function 36
4.2 Cost Function 42
4.3 Feature Selection 43
4.4 Data Filtering 44
4.5 Metrics to Evaluate Machine Learning Algorithms 44

5 Simulation and Results of Cantilever Beams with Inelastic Materials and Elasto-Plastic Shell Structures **49**
5.1 Cantilever Beams with Inelastic Materials 49
5.2 Elasto-plastic Shell Structures 65

6 Simulation and Results of Car Collisions **81**
6.1 Data Collection 81
6.2 Animation Results 88
6.3 Discussions on Metrics to Evaluate Machine Learning Algorithms 88
6.4 Discussions on Activation Functions 95
6.5 Discussions on Cost Functions 99
6.6 Discussions on Data Filtering 99

7 Closing **100**
7.1 Summary 100
7.2 Future Work 102
7.3 Broader Impact of the Dissertation 103

Bibliography **105**

List of Figures

3.1 SVM process overview 11
3.2 Decision tree simplified 12
3.3 Random forest simplified 13
3.4 The processes of passing information in neurons [23] 14
3.5 Flowchart of developing the machine learning neural network 19
3.6 Illustration of neuron structure of the neural network 21
3.7 Graphical illustration of bias and variance [32] 23
3.8 Dropout in neural network 24
3.9 Illustration of machine learning approach to identify structure failure load conditions. . 26
3.10 Impact load identification method 28
3.11 Flowchart of data processing 29
3.12 Illustration of structure of the neural network 33

4.1 Relu function plot 37
4.2 Sigmoid function plot 38
4.3 Tanh function plot 39
4.4 Softmax function plot 40
4.5 Relu Square function plot 41
4.6 Overview of k-fold cross-validation method [22] 45

5.1 Finite element model of the cantilever beam and the loading positions 51
5.2 Dynamic loading time history 51
5.3 Lastic displacement distribution: (a) Plastic displacement along horizontal direction, and (b) Plastic displacement along vertical direction. 52
5.4 Permanent plastic strain distribution: (a) Plastic strain component ϵ_{11}^{p}, (b) Plastic strain component ϵ_{22}^{p}, and (c) Plastic strain component ϵ_{12}^{p}. 54
5.5 Collect one set of training data input 55
5.6 Training loss for prediction of the static loads on training nodes 57
5.7 Training and testing data 59
5.8 Training loss for prediction of the impact loads on training nodes 60
5.9 Predicted result of the loads on the nodes 61
5.10 Training and testing data 62

5.11 Predicted result of the loads in the intervals . . . 63
5.12 Refined interval . . . 63
5.13 Finite element contact collision model of a hemispherical shell and a rigid cylindrical body . . . 66
5.14 Velocity-time curve for the static and dynamic analysis . . . 68
5.15 Different initial longitude and latitude position of the rigid cylindrical body . . . 69
5.16 Training loss of the duration of dynamic cases . . . 70
5.17 Training loss of the location of dynamic cases . . . 70
5.18 Training loss of the location of static cases . . . 71
5.19 Predicted location of static and dynamic loads on training points . . . 73
5.20 Predicted results of quasi-static speeds . . . 74
5.21 Predicted results of dynamic speeds and duration . . . 75
5.22 Predicted locations in interval . . . 76
5.23 Stress distribution of Dynamics case 2: (a) stress distribution of test data; (b) recovered stress distribution generated by ML results. . . . 77
5.24 Stress distribution of Dynamics case 4: (a) stress distribution of test data; (b) recovered stress distribution generated by ML results. . . . 78
5.25 Symmetry properties of hemisphere shell and its finite element mesh . . . 78
5.26 Deformation transferring process . . . 79

6.1 FEM model of a car . . . 81
6.2 FEM models of two cars . . . 82
6.3 FEM models of two cars in LS-DYNA software interface . . . 83
6.4 Offset of two crashed cars . . . 84
6.5 Velocities of two crashed cars . . . 84
6.6 Angle between two crashed cars . . . 85
6.7 Strain-stress curve of the SPCEN steel . . . 85
6.8 Strain-stress curve of the thixocast A356 alloy . . . 86
6.9 Residual displacement of a crash car after collision . . . 87
6.10 Animation of car collisions with zero angle (time: $0.04s$ - $0.2s$) . . . 89
6.11 Strain of the left car after car collision with zero angle . . . 90
6.12 Animation of car collisions with non-zero angle (time: $0.04s$ - $0.2s$) . . . 91
6.13 Strain of the left car after car collision with non-zero angle . . . 92
6.14 Animation comparison between the original case and predicted case after car collisions 93
6.15 Comparison of the residual displacement of the left car between the original case and predicted case after car collisions . . . 94
6.16 Comparison of the residual displacement of the right car between the original case and predicted case after car collisions . . . 95
6.17 The derivative of Relu function plot . . . 97
6.18 The derivative of Tanh function plot . . . 97
6.19 The derivative of Sigmoid function plot . . . 98
6.20 The derivative of Relu Square function plot . . . 98

List of Tables

3.1 Hyper-parameters of the DNN 32

5.1 Mechanical properties of AISI 4340 steel (33 HRc) (From [47, 63]). 50
5.2 Loads for the static analysis 52
5.3 Loads for the dynamic analysis 52
5.4 The correct and predicted loads of the testing cases $(10^7 N)$ 56
5.5 Predicted errors of the testing cases (%) 56
5.6 Correct values of the testing data 58
5.7 Predicted results of the testing data 58
5.8 Predicted errors of the testing data (%) 58
5.9 Predicted errors of the nodal impact loads(%) 59
5.10 Predicted errors of the loads in intervals(%) 64
5.11 Correct values of the predicted impact loads 64
5.12 Predicted value of the impact loads 64
5.13 Predicted errors of the nodal impact loads(%) 65
5.14 Material parameters of the elasto-plastic shell structures 67
5.15 Loads and duration for the static and dynamic analysis 68
5.16 Information of testing cases on training points 71
5.17 Architecture and hyper-parameters of the neural network 72
5.18 Predicted errors of loads acting on training points 72
5.19 Information of testing cases in interval 74
5.20 Predicted errors of loads acting in interval 75
5.21 Predicted results of the impact on $(180°, 55°)$ 80

6.1 Geometry information of the model 82
6.2 Johnson-Cook model parameters 86
6.3 Initial speed combinations (km/h) 87
6.4 Results of offset 90
6.5 Results of velocities 92
6.6 Results of angles 93
6.7 Comparison of different activation functions 96
6.8 Comparison of different cost functions 99

Acknowledgments

I would like to sincerely thank the following people, without whose help and patience this dissertation would not have been possible:

My advisor: Professor Shaofan Li.

Members of my dissertation committee: Professor Khalid M. Mosalam and Professor Lin Lin.

My co-authors: Guorong Chen, Qijun Chen, Shaofei Ren, Qingsong Tu, Chao Wang, Ao Deng, Kevin Zhu and Yifei Liu.

Various collaborators, visiting scholars, alumni, and friends of Professor Li's research group.

Chapter 1

Introduction

1.1 Background

Big Data Era

The integration of computer technology into science and daily life has enabled the collection of massive volumes of data, such as high-throughput biological assay data, climate data, website transaction logs, and credit card records. The concept of big data has been around for years; most organizations now understand that if they capture all the data that streams into their businesses, they can apply analytics and get significant value from it. But even in the 1950s, decades before anyone uttered the term big data, people were using basic analytics (essentially numbers in a spreadsheet that were manually examined) to uncover insights and trends.

The new benefits that big data analytics brings to the table, however, are speed and efficiency. Whereas a few years ago a business would have gathered information, run analytics and unearthed information that could be used for future decisions, today we can identify insights for immediate decisions. The ability to work faster and stay agile gives organizations a competitive edge they did not have before [79, 45].

Deep Learning is especially very popular. There are three main reasons for that:

One of the things that increased the popularity of Deep Learning is the massive amount of data that is available in the current year, which has been gathered over the last years and decades. This enables Neural Networks to really show their potential since they get better the more data fed into them. In comparison, traditional Machine Learning algorithms will certainly reach a level, where more data doesnt improve their performance.

Another very important reason is the computational power that is available nowadays, which enables processing more data. The computational power is multiplied by a constant factor for each unit of time (e.g., doubling every year) rather than just being added to incrementally. This means

that computational power is increasing exponentially [78].

The third factor that increased the popularity of Deep Learning is the advances that have been made in the algorithms itself. These recent breakthroughs in the development of algorithms are mostly due to making them run much faster than before, which makes it possible to use more and more data.

Computer Science Methods Application

Computer science methods are widely applied to different fields: urban complex systems, computational finance, computational biology, complex systems theory, engineering and so on. In todays financial markets huge volumes of interdependent assets are traded by a large number of interacting market participants in different locations and time zones. Their behavior is of unprecedented complexity and the characterization and measurement of the risk inherent to these highly diverse set of instruments is typically based on complicated mathematical and computational models. Using information theory, non-equilibrium dynamics and explicit simulations computational systems theory tries to uncover the true nature of complex adaptive systems.

Computational science and engineering (CSE) is a relatively new discipline that deals with the development and application of computational models and simulations, often coupled with high-performance computing, to solve complex physical problems arising in engineering analysis and design (computational engineering) as well as natural phenomena (computational science) [38]. CSE has been described as the ”third mode of discovery” (next to theory and experimentation). In many fields, computer simulation is integral and therefore essential to business and research. Computer simulation provides the capability to enter fields that are either inaccessible to traditional experimentation or where carrying out traditional empirical inquiries is prohibitively expensive. CSE should neither be confused with pure computer science, nor with computer engineering, although a wide domain in the former is used in CSE (e.g., certain algorithms, data structures, parallel programming, high performance computing) and some problems in the latter can be modeled and solved with CSE methods.

Computational Mechanics

Computational mechanics is the discipline concerned with the use of computational methods to study phenomena governed by the principles of mechanics. In granular mechanics, macroscopic approaches treat a granular material as an equivalent continuum at macro-scale, and study its constitutive relationship between macro-quantities, such as stresses and strains. On the other hand, microscopic approaches consider a granular material as an assembly of individual particles interacting with each other at micro-scale (i.e., particle-scale), and the physical quantities under study are forces and displacements [4].

Computational mechanics encompasses the development and use of computational methods for studying problems governed by the laws of mechanics. Modern computational mechanics is embodied in the broad field of computational science and engineering. This discipline plays a fundamental role in a vast number of many important problems in science and engineering such as aircraft design, drug delivery, crashworthiness, materials design, tissue engineering, biomedical imaging, prediction of natural events (e.g., climate modeling), energy exploration and exploitation, and others.

There are not too much work related to the application of computational science methods on the computational mechanics domain. Thus, following this computer science methods trend, it is timely to implement the technologies in computational mechanical applications and evaluate its potential benefits.

1.2 Thesis Organization

A novel deep learning computational framework is developed in this dissertation to determine and identify the damage load conditions of different types of structures, including cantilever beams of inelastic materials, elasto-plastic shell structures of inelastic materials and car collisions subjected to mechanical forcing actions. The organization of this thesis is as follows.

The introduction and motivation are stated in Chapter 1. After that, Chapter 2 first provides descriptions of the previous related work of machine learning methodology, followed by a discussion of its previous application, which is the fundamental background of this work. Then, the history of inverse problem of computational mechanics is discussed in detail. The last section in Chapter 2 focuses on our problem statement.

Chapter 3 provides a description and summary of general machine learning methods, as well as overview of the unsupervised algorithms and the supervised algorithms, along with the comparison between different methods. Then we state the methods we use in our work, which contains the full description of all methodologies, such as data filtering, principal component analysis, etc., and parameters, such as numbers of hidden layers, numbers of neurons in each layer, etc., of our neural network models.

Based on the methods we use in Chapter 3, we continue to extend our study on the theoretical part of machine learning methodology in Chapter 4. General study of loss functions and activations functions are summarized first. Then studies are conducted upon different cost functions, activation function and some other factors of our machine learning algorithm.

In Chapter 5, two simulations are conducted using the developed novel machine (deep) learning computational framework to determine and identify damage loading parameters (conditions) for structures and materials based on the permanent or residual plastic deformation distribution or

damage state of the beam and shell structure. We first describe how the models are built on, as well as the data collection process. Then we report all the simulation and numerical results in this chapter, and followed by discussions and analysis of the results.

Chapter 6 continues to develop a novel deep learning computational framework to determine and identify the damage load conditions for car collisions.The process of building models on and the data collection are first introduced, followed by all the simulation and numerical results. Then discussions and analysis of the results are performed, including the discussions of different activation functions and cost functions. A new metric to evaluate machine learning algorithm, L^2 norm method, is discussed and used in the results analysis.

We close the dissertation by summarizing our results, analysis and discussions, outlining future work and reflecting on the broader implications of the dissertation in Chapter 7.

Chapter 2

Related Work and Problem Statement

This chapter discusses the related work and problem statement. It first introduces the related work of machine learning methodology followed by a discussion of previous applications, which is the fundamental background of this work. Then, the history of inverse problem of computational mechanics is discussed. The last section focuses on highlighting our problem statement.

2.1 Summary of Related Machine Learning Work

Machine learning has a comparatively short history. In 1950 Alan Turing published Computing Machinery and Intelligence, in which he asked: "Can machines think?", which is a question that we still wrestle with [86]. Marvin Minsky and Dean Edmonds built the first artificial neural network, a computer-based simulation of the way organic brains work the next year. During the 1950s to 1960s there was enormous enthusiasm for AI research.

The idea of a computer which programs itself was very appealing in 1960s. But it was very difficult to implement. Von Neumann architecture was gaining in popularity. In 1972, Kohonen and Anderson developed a similar network independently of one another. They both used matrix mathematics to describe their ideas but did not realize that what they were doing was creating an array of analog ADALINE circuits. The neurons are supposed to activate a set of outputs instead of just one. The first multilayered network was developed in 1975, an unsupervised network.

Then, work on machine learning shifted from a knowledge-driven approach to a data-driven approach. Scientists began creating programs for computers to analyze large amounts of data and draw conclusions or learn from the results. Support vector machines and recurrent neural networks became popular. The fields of computational complexity via neural networks and super-Turing computation started. A lot of core techniques were developed, such as backpropagation [49, 25] and deepmind [112, 108]. Artificial intelligence and machine learning technologies are developing rapidly nowadays, especially in applications of deep learning in computer vision, which made giant progress in recent years [1]. In addition, the objective of implementation of machine learning

is to make computers perform labor-intensive repetitive tasks and also learn from past experiences.

Until now, there are combinations of mechanics and machine learning works these years, such as [41, 103, 72, 40].

The costs of fatalities and injuries due to traffic accidents have a great impact on the society. In recent years, researchers have paid increasing attention to determining factors that significantly affect severity of driver injuries caused by traffic accidents such as [70, 19]. There are several approaches that researchers have employed to study this problem. These include neural network and other machine learning algorithms.

Virtual product development based on numerical simulation is nowadays an essential tool in the computational mechanics, especially car industry. It is used to analyze the influence of design parameters on the weight, costs, functional properties, etc. of new car models [53]. Automobile mechanical engineers spend a considerable amount of their time analyzing these influences by inspecting the arising simulations one at a time. There exist researches focus on methods from machine learning to semi-automatically analyze the arising finite element data and thereby significantly assist in the overall engineering process [10, 18].

2.2 Inverse Problem in Computational Mechanics

There are a number of ways in which investigators review and evaluate the causes and circumstances of a car accident. In the event of a complicated accident involving a serious injury or death, a specialized reconstructionist may be needed to analyze the crash site and to identify the cause of the collision. These experts help determine the cause of a crash and help explain how the crash could have been prevented. Their observations and conclusions can prove useful in court and they can lead to changes in the way highways are maintained and vehicles are designed.

All of the engineering materials, products and structures are designed, manufactured or constructed with an intention to function properly. However, they can fail, get damaged or may not operate or function as intended due to various reasons including material or design flaws, extreme loading, etc. It is important to identify the reasons of these failures or damage situations to improve the designs and detect any flaws in the materials or designs. One of the essential requirements of identifying the reasons of these failures is to know the loading conditions that lead to the failures.

Unfortunately, these loading conditions are not readily known while the forensic signatures such as the plastic strains or plastic deformations are easily measurable. For example, in car crashes, the impact loads on cars are not known, while the permanent deformations can be quantified after the fact. If the impact loads can be determined, it could potentially help insurance companies determine which party is responsible for the accident, and help car manufactures develop more realistic crash test scenarios. Both of these have high potential for concrete and significant economic

impact.

What emerges from these considerations is an inverse problem of finding loading conditions from engineering responses. This represents an inverse of current engineering practice, in which the typical setup is to develop finite element models of structures, subject them to static and dynamic loading conditions, and then compute the resulting strains and residual displacements.

For the inverse problem, we believe machine learning and artificial intelligence techniques provide an effective solution. Machine learning and artificial intelligence encompass powerful tools for extracting complicated relationship between input and output sampling data, potentially through a training process, and then using the uncovered relationship to make predictions [52, 92]. Machine learning and artificial intelligence have found a large number of successful applications in various fields beyond their birthplace in computer science [109, 11, 106]. Recent years have also witnessed a number of studies devoted to applying machine learning techniques to explore forensic materials engineering problems [64, 91]. In the context of this chapter, the measurable engineering responses would be fed as input, with the loading conditions as output. The inverse nature of the problem does not hinder machine learning and artificial intelligence's effectiveness in discovering complex mathematical relationships between input and output.

2.3 Problem statement

Our goal is to develop a novel machine (deep) learning computational framework to solve the inverse problem by determining and identifying damage loading parameters (conditions) for structures and materials based on the engineering responses such as permanent or residual plastic deformation distribution or damage state of the structure.

This work will combine the current mature state of finite element models with the recent advances in machine learning methodologies. This approach will advance the state of the art in forensic materials engineering, which seeks to examine material evidence and determine the original causes [82, 132, 133, 73]. We believe the machine learning based approach can solve many previously intractable problems, with prior approaches incurring impractical computational costs due to the scale of the finite element models, the large degrees of freedom, and the complex and dynamic nature of the loading forces.

We want to begin with a detailed description of the machine learning algorithm used, then outline our process for gathering the required data to train the machine learning algorithms. This will be followed by examples that demonstrate how we solve the inverse problem, including a cantilever beam of inelastic materials statically loaded at different locations, and the same beam loaded dynamically with impact loading, elasto-plastic shell structures and car collision examples. We seek to demonstrate with these examples that the machine learning algorithms can accurately

identify both static loading and impact loading conditions based on observed residual plastic strain, deformation or other features.

Chapter 3

Machine Learning Methodology

3.1 General Machine Learning Methodology

Machine learning is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as ”training data”, in order to make predictions or decisions without being explicitly programmed to perform the task [76, 125]. Machine learning algorithms are used in a wide variety of applications, such as email filtering, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning [20, 36]. In its application across business problems, machine learning is also referred to as predictive analytics.

Data scientists use many different kinds of machine learning algorithms to discover patterns in big data that lead to actionable insights. At a high level, these different algorithms can be classified into two groups based on the way they learn about data to make predictions: supervised and unsupervised learning.

Supervised Learning Methods

Supervised machine learning is the more commonly used between the two. It includes such algorithms as linear and logistic regression, multi-class classification, and support vector machines. Supervised learning is named as such because the data scientist acts as a guide to teach the algorithm what conclusions it should come up with. Supervised learning requires that the algorithms possible outputs are already known and that the data used to train the algorithm is already labeled with correct answers. Supervised learning problems can be further grouped into regression and

classification problems [13, 115, 62, 104].

Classification: A classification problem is when the output variable is a category, such as red or blue or disease and no disease. For example, a classification algorithm will learn to identify animals after being trained on a dataset of images that are properly labeled with the species of the animal and some identifying characteristics.

Regression: A regression problem is when the output variable is a real value, such as dollars or weight. Some common types of problems built on top of classification and regression include recommendation and time series prediction respectively.

Some popular examples of supervised machine learning algorithms are: linear regression for regression problems, random forest for classification and regression problems and support vector machines for classification problems.

Unsupervised Learning Methods

Unsupervised learning is where there is only input data and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning above, there are no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data. Unsupervised machine learning is more closely aligned with what some call true artificial intelligence the idea that a computer can learn to identify complex processes and patterns without a human to provide guidance along the way. Although unsupervised learning is prohibitively complex for some simpler enterprise use cases, it opens the doors to solving problems that humans normally would not tackle [55, 122].

Unsupervised learning problems can be further grouped into clustering and association problems.

Clustering: A clustering problem is where the objective to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

Association: An association rule learning problem is where the aim is to discover rules that describe large portions of data, such as people that buy X also tend to buy Y.

Some popular examples of unsupervised learning algorithms are: k-means for clustering problems, principal and independent component analysis, and algorithm for association rule learning problems.

Model Selection

Model selection is the task of selecting a statistical model from a set of candidate models, given data. In the simplest cases, a pre-existing set of data is considered. However, the task can also involve the design of experiments such that the data collected is well-suited to the problem of model selection. Given candidate models of similar predictive or explanatory power, the simplest model is most likely to be the best choice. Choosing to use either a supervised or unsupervised machine learning algorithm typically depends on factors related to the structure and volume of data and the use case of the issue at hand. In our problem, supervised machine learning algorithm is chosen to solve the problem based on the structure and volume of our data [2, 3].

Our problem could be seen as a classification problem or a regression problem. Thus, there exist a lot of machine learning models we can choose. The main four models we tried are support-vector machine, decision tree, random forest and artificial neural network.

Support-Vector Machines

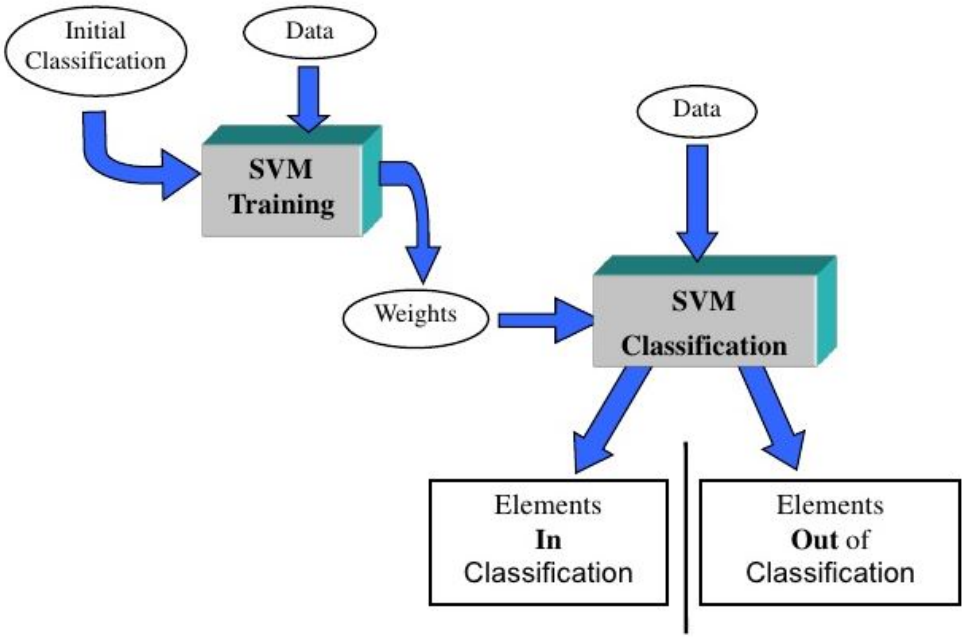


Figure 3.1: SVM process overview

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis, as shown in Fig. 3.1. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that

assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall [131, 58].

In addition to perform linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Decision Tree

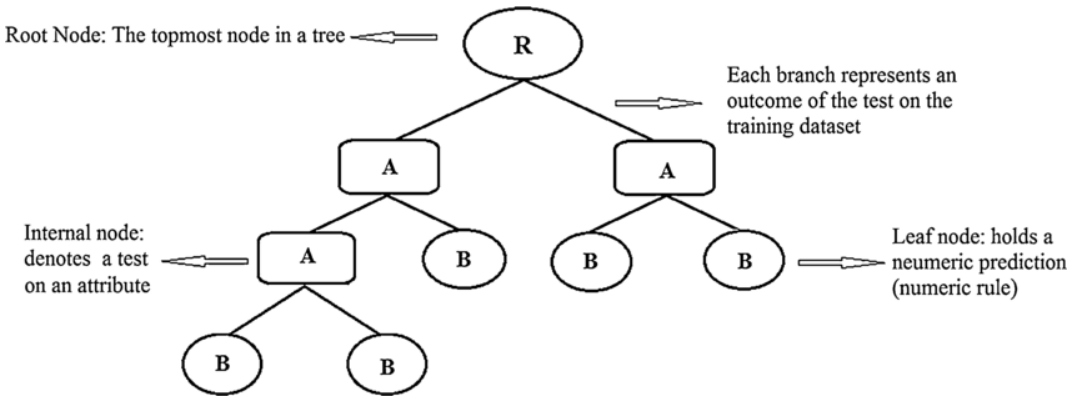


Figure 3.2: Decision tree simplified

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements [67, 69]. The simplified decision tree is shown in Fig. 3.2.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

A decision tree is a flowchart-like structure in which each internal node represents a ”test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

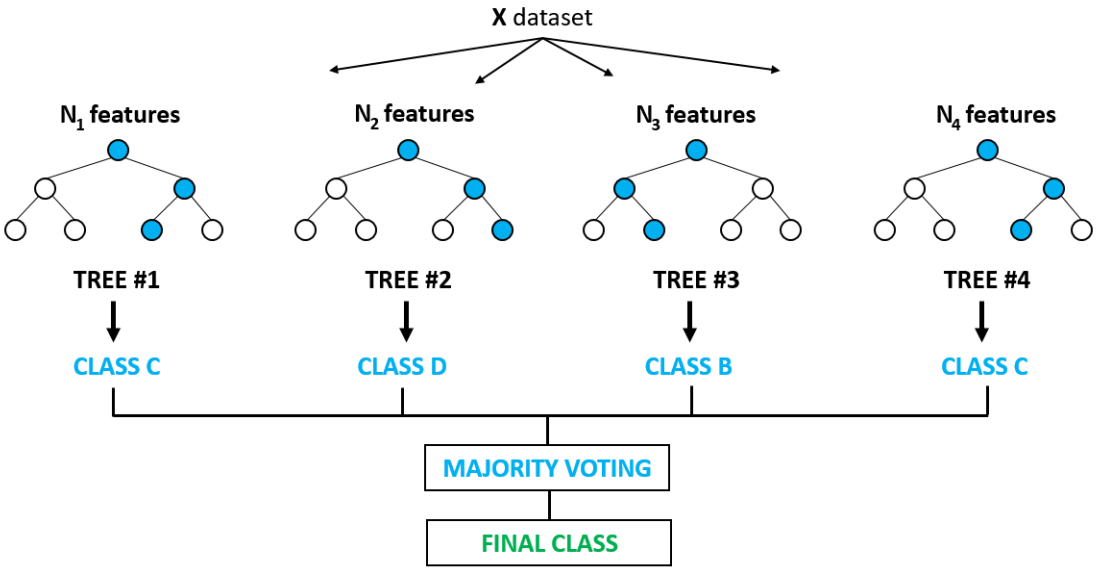


Figure 3.3: Random forest simplified

Random Forests

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. The first algorithm for random decision forests was created by using the random subspace method, which is a way to implement the "stochastic discrimination" approach to classification [56, 6]. The simplified random forest is shown in Fig. 3.3.

Decision trees are a popular method for various machine learning tasks. Tree learning come closest to meeting the requirements for serving as an off-the-shelf procedure for data mining because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate.

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

Artificial Neural Network

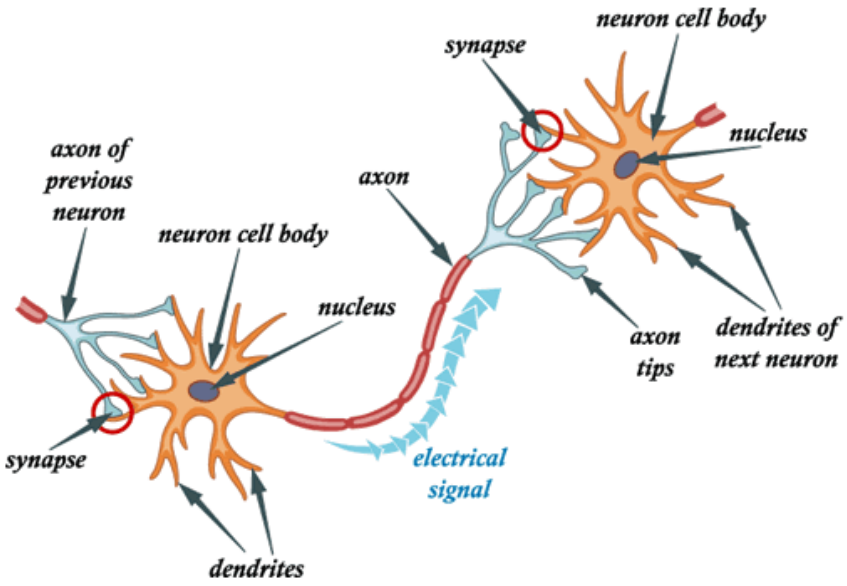


Figure 3.4: The processes of passing information in neurons [23]

In the human brain, there are billions of cells called neurons, which processes information in the form of electric signals. External information/stimuli is received by the dendrites of the neuron, processed in the neuron cell body, converted to an output and passed through the Axon to the next neuron, as shown in Fig. 3.4. The next neuron can choose to either accept it or reject it depending on the strength of the signal.

An ANN is a very simplistic representation of a how a brain neuron works. Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems ”learn” to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as ”cat” or ”no cat” and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process [43].

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives

a signal can process it and then signal additional artificial neurons connected to it.

In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

Pros and Cons of Different Machine Learning Models

The machine learning algorithm that should be used for a given problem depends on the size, quality, and nature of the data. It also depends on what the answer wants to be used for and how the math of the algorithm was translated into instructions for the utilized computer. Besides, it depends on the computational resources available since different algorithm will lead to different requirement related to computational time. Even the most experienced data scientists can't tell which algorithm will perform best before trying them.

Support-Vector Machines

Support-Vector Machines has a regularization parameter, which makes the user think about avoiding over-fitting. It uses the kernel trick, so that one can build in expert knowledge about the problem via engineering the kernel. Also, an SVM is defined by a convex optimization problem (no local minima) for which there are efficient methods. Lastly, it is an approximation to a bound on the test error rate, and there is a substantial body of theory behind it which suggests it should be a good idea.

However, there still some problems that exist for SVM. The disadvantages are that SVM only really covers the determination of the parameters for a given value of the regularization and kernel parameters and choice of kernel. In a way the SVM moves the problem of over-fitting from optimizing the parameters to model selection. Thus, the main disadvantage of the SVM algorithm is that it has several key parameters that need to be set correctly to achieve the best classification results for any given problem. Parameters that may result in an excellent classification accuracy for

problem A, may result in a poor classification accuracy for problem B. The lack of generalization of SVM can't satisfy the objectives of this work.

Decision Tree

For decision tree method, there are several advantages. One big advantage of the decision tree model is its transparent nature. Unlike other decision-making models, the decision tree makes explicit all possible alternatives and traces each alternative to its conclusion in a single view, allowing for easy comparison among the various alternatives. A major decision tree analysis advantages is its ability to assign specific values to problem, decisions, and outcomes of each decision. This reduces ambiguity in decision-making. Every possible scenario from a decision finds representation by a clear fork and node, enabling viewing all possible solutions clearly in a single view [29, 105, 33].

The decision tree is the best predictive model as it allows for a comprehensive analysis of the consequences of each possible decision, such as what the decision leads to, whether it ends in uncertainty or a definite conclusion, or whether it leads to new issues for which the process needs repetition. Decision trees also score in ease of use. The decision tree provides a graphical illustration of the problem and various alternatives in a simple and easy to understand format that requires no explanation. Unlike other decision-making tools that require comprehensive quantitative data, decision trees remain flexible to handle items with a mixture of real-valued and categorical features, and items with some missing features. Once constructed, they classify new items quickly. Another of the decision tree analysis advantages is that it focuses on the relationship among various events and thereby, replicates the natural course of events, and as such, remains robust with little scope for errors, provided the inputted data is correct. A decision tree finds use to make quantitative analysis of business problems, and to validate results of statistical tests. It naturally supports classification problems with more than two classes and by modification, handles regression problems [119].

However, there are still several disadvantages based on the data type and feature size in this study. The reliability of the information in the decision tree depends on feeding the precise internal and external information at the onset. Even a small change in input data can at times, cause large changes in the tree. Changing variables, excluding duplication information, or altering the sequence midway can lead to major changes and might possibly require redrawing the tree. Another fundamental flaw of the decision tree analysis is that the decisions contained in the decision tree are based on expectations, and irrational expectations can lead to flaws and errors in the decision tree. Although the decision tree follows a natural course of events by tracing relationships between events, it may not be possible to plan for all contingencies that arise from a decision, and such oversights can lead to bad decisions. Decision trees are easy to use compared to other decision-making models, but preparing decision trees, especially large ones with many branches, are complex and time-consuming. Since the data size of this study is significantly large, it is not

possible to afford the computational time that decision tree method needs [93, 34].

Besides, one of the decision tree advantages are its listing comprehensive information and all possible solutions to an issue. Such comprehensiveness can, however, work both ways and need not always be an advantage. The most significant dangers with such excessive information are ”paralysis of analysis” where the decision makers burdened with information overload takes time to process information, slowing down decision-making capacity. The time spent on analysis of various routes and sub routes of the decision trees would find better use by adopting the most apparent course of action straightway and getting on with the core business process, making such information rank among the major disadvantages of a decision tree analysis [54]. In this study, the size of feature is defined as the number of mesh nodes, which is a huge number. That will provide too much information to decision tree. Thus, decision tree is not suitable for this study and other methods are considered.

Random Forests

Random forests overcome several problems with decision trees, including: Reduction in overfitting: by averaging several trees, there is a significantly lower risk of overfitting; Less variance: By using multiple trees, the chance of stumbling across a classifier that doesnt perform well is reduced because of the relationship between the train and test data. As a consequence, in almost all cases, random forests are more accurate than decision trees [120, 12, 35].

The main disadvantage of random forests method is their complexity, which is same with decision tree method. They are much harder and time-consuming to construct than decision trees. They also require more computational resources and are also less intuitive. When having a large collection of decision trees it is hard to have an intuitive grasp of the relationship existing in the input data. In addition, the prediction process using random forests is time-consuming than other algorithms [94, 56].

A large number of trees can make the algorithm to slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough. But in this study, run-time performance is important, accordingly other approaches need to be preferred.

Besides, Random Forest is a predictive modeling tool and not a descriptive tool. Because description of the relationships in the data is sought in this study, other approaches are preferred [27].

Artificial Neural Network

ANNs have some key advantages that make them most suitable for certain problems and situations, such as this study:

1. ANNs have the ability to learn and model non-linear and complex relationships, which is really important because many of the relationships between inputs and outputs are non-linear as well as complex in this study. Nonlinear systems have the capability of finding shortcuts to reach computationally expensive solutions. These systems can also infer connections between data points, rather than waiting for records in a data source to be explicitly linked. This nonlinear short-cut mechanism is fed into artificial neural networking, which makes it valuable in commercial big-data analysis.

2. ANNs are able to generalize. After learning from the initial inputs and their relationships, it can infer unseen relationships on unseen data as well, thus making the model generalize and predict on unseen data. Neural networks can learn organically. This means an artificial neural network's outputs aren't limited entirely by inputs and results given to them initially by an expert system. Artificial neural networks have the ability to generalize their inputs. This ability is valuable for robotics and pattern recognition systems [87].

3. Unlike many other prediction techniques, ANN does not impose any restrictions on the input variables, like how they should be distributed. Additionally, many studies [81, 26, 90, 30] have shown that ANNs can better model heteroskedasticity i.e. data with high volatility and non-constant variance, given its ability to learn hidden relationships in the data without imposing any fixed relationships in the data. This is something very useful in this study where data volatility in terms of time is very high.

4. Artificial neural networks have the potential for high fault tolerance. When these networks are scaled across multiple machines and multiple servers, they are able to route around missing data or servers and nodes that can't communicate.

5. Artificial neural networks can do more than routing around parts of the network that no longer operate. If they are asked for finding out specific data that is no longer communicating, these artificial neural networks can regenerate large amounts of data by inference and help in determining the node that is not working. This trait is useful for networks that require informing their users about the current state of the network and effectively results in a self-debugging and diagnosing network [124].

In summary, artificial neural network is a nonlinear model that is easy to use and understand compared to statistical methods. Artificial neural network is non-parametric model while most of statistical methods are parametric model that need higher background of statistics. Artificial neural network with Back propagation (BP) learning algorithm is widely used in solving various

classification and forecasting problems, even though BP convergence is slow but it is guaranteed. Besides, ANN is a black box learning approach. The interpretability of relationship between input and output is comparatively limited. The data of this study are non-linear and input and output have complex relationships.

Based on the four potential methods, artificial neural network is the most accurate method for this study with a reasonable computation time.

3.2 Methodology for Cantilever Beams with Inelastic Materials

Deep Neural Network Model

Recently, machine learning, especially deep neural networks, is one of the most popular key words in every scientific or engineering field. Deep learning architectures, such as deep neural networks, deep belief networks and recurrent neural networks, have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design and board game programs, where they have produced results comparable to and in some cases superior to human experts, e.g. [21, 77].

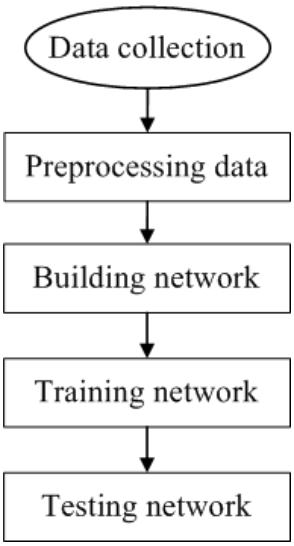


Figure 3.5: Flowchart of developing the machine learning neural network

Using machine learning methods, one could make relatively accurate predictions on some problems that were difficult to be solved before. In this work, we adopted a fundamental model of deep

neural network (DNN), namely an artificial neural network. There are five stages in our model, as shown in Fig 3.5.

(1) Data collection: After building up model in Abaqus and performing simulations, collect data from software Abaqus.

(2) Preprocessing data: Data cleaning and processing with feature selection and feature engineering. During this stage, we applied dimension reduction, removed irrelevant data, and created new features so the model could perform better.

The original data of static loading condition has 120 variables, which indicate 60 pairs of displace from x and y directions. The 60 pairs of data represent three parts of a cantilever beam (explained in detail in the next section) which are the top, middle, and bottom parts. Due to the similarities, the top and bottom data was eliminated and we only keep the middle parts. Since the cantilever beam has a relatively small displacement on x-direction, we only keep the y displacement to simplify the model. When doing the feature engineering, we tried to use our expertise and mathematical methods to create new relative features and established five new variables, including finding slope from plotting data of displacement x and displacement y, the summation of displacement y, the amplitude of displacement x, and the centroid distance. (First, we find a linear fit for the deformation change, namely, the approximate slope of the displacement x and displacement y from plotting data. Second, we generate the sum of displacement of y as another new feature. Third, to make the feature more obvious, we create a variable by using the top displacement x minus the bottom displacement x. Last but not least, we made centroid distance as a new feature. The equation is

$$centroid\ distance = \sqrt{(u_x - center)^2 + (u_y - center)^2)} \tag{3.1}$$

where ux is the displacement of x and uy is the displacement of y) For the dynamic data, we did a relative radical method. Like the static data we pick only 20 variables, instead of keep them we use the product of displacement x with displacement y, which lead all variables new. Then we applied summation of the variable and the centroid distance of the new dataset.

(3) Building network: Initialize bias, weight, number of layers and number of neurons in each layer.

(4) Training network: Application of deep neural network to obtain a specific mathematical model. It is noted that the basic mathematical model for the first stage of neural network is a simplified projection pursuit regression [37],

$$\hat{y_i} = \sum_{j=1}^{n} g_j(w_j^T x_i) \tag{3.2}$$

where g(x) is an activation function, w is a distributed weight, and x is an observation. During the second stage, the w is redistributed to optimize the loss or error used through back-propagation.

(5) Testing network: Using a chosen test data set, which is different than the training sets, to validate the model and analyze errors.

Models and Settings

As mentioned earlier, a deep neural network [80] is used in this study to solve this inverse problem with procedures considered for the features. One sample fully-connected layer is shown in Fig 3.6 [14]. In one certain neuron j, the employed mathematical model is described as

$$g_j(\beta_k^T X, \beta_0) = \sigma(\beta_0 X_0 + \beta_k^T X), \quad where\ X_0 = 1 \tag{3.3}$$

One can clearly observe from Fig 3.6 that, the input data flow(X) multiplied by the distributed weight(β_k) and then added a bias (β_0) as a parameter of an activation function ($\sigma(x)$), reflects the mathematical model in equation (3.3). Among all the options of activation functions, such as hyperbolic $tanh$, Sigmoid, or Relu. According to Ramachandran, "currently, the most successful and widely-used activation function is the Rectified Linear Unit." [100] In our case, Rectified Linear Unit performs well as an activation function $\sigma(x)$ for both static and dynamic loading conditions. ReLU provides a nonlinear function $f(x) = max(0, x)$. Over-fitting is usually a concern that a model would perform too "well" on the training data set. In order to prevent over-fitting, a dropout was applied during the training process, "the key idea is to randomly drop units (along with their connections) from the neural network during training." [118]

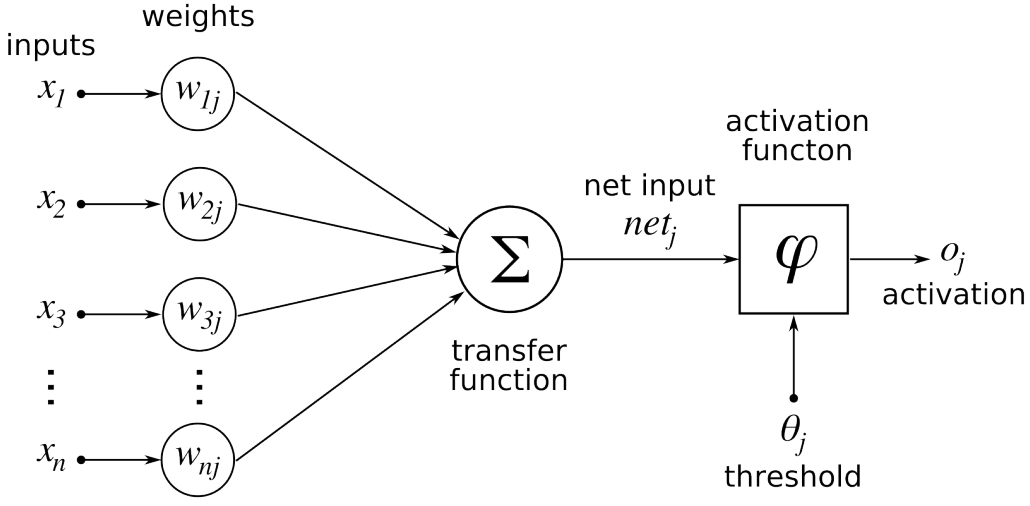


Figure 3.6: Illustration of neuron structure of the neural network

Further, the used loss function is the Mean Square Error (MSE), defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{3.4}$$

In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errorsthat is, the average squared difference between the estimated values and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss. The fact that MSE is almost always strictly positive (and not zero) is because of randomness or because the estimator does not account for information that could produce a more accurate estimate.

The MSE is a measure of the quality of an estimatorit is always non-negative, and values closer to zero are better.

The MSE is the second moment (about the origin) of the error, and thus incorporates both the variance of the estimator (how widely spread the estimates are from one data sample to another) and its bias (how far off the average estimated value is from the truth). For an unbiased estimator, the MSE is the variance of the estimator. Like the variance, MSE has the same units of measurement as the square of the quantity being estimated. In an analogy to standard deviation, taking the square root of MSE yields the root-mean-square error or root-mean-square deviation (RMSE or RMSD), which has the same units as the quantity being estimated; for an unbiased estimator, the RMSE is the square root of the variance, known as the standard error.

MSE is commonly adopted as a loss function in statistical models. It provides an intuitive measurement of errors. The objective is to minimize MSE and make the model fit both the training data and validation data well. An optimizer will adjust between forward and backward propagation in order to achieve this objective. The optimization tool used herein to minimize MSE is the Adam optimizer [71] with exponential descent in learning rate.

Bias-variance Tradeoff

In statistics and machine learning, the bias-variance tradeoff is the property of a set of predictive models whereby models with a lower bias in parameter estimation have a higher variance of the parameter estimates across samples, and vice versa. The biasvariance dilemma or problem is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set.

The bias is an error from erroneous assumptions in the learning algorithm. The error due to bias is taken as the difference between the expected (or average) prediction the our model and the correct value which is to be predicted. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting). The variance is an error from sensitivity

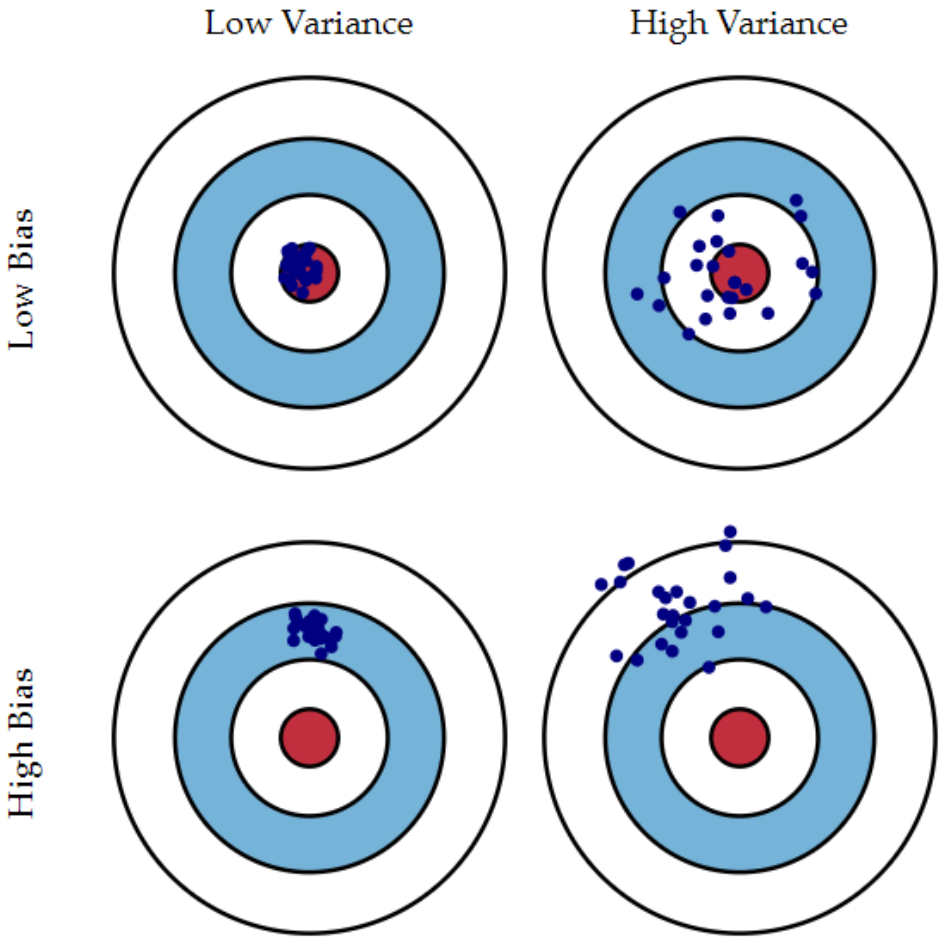


Figure 3.7: Graphical illustration of bias and variance [32]

to small fluctuations in the training set. The error due to variance is taken as the variability of a model prediction for a given data point. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting), as shown in Fig. 3.7. The biasvariance decomposition is a way of analyzing a learning algorithm's expected generalization error with respect to a particular problem as a sum of three terms, the bias, variance, and a quantity called the irreducible error, resulting from noise in the problem itself [42, 107].

In summary, bias and variance are two important parameters to tell if the machine learning model is good or not. Different conditions of combinations of bias and variance may lead to different problems: overfitting and underfitting, and need different solutions. This tradeoff applies to all forms of supervised learning, including this regression study.

Drop-out

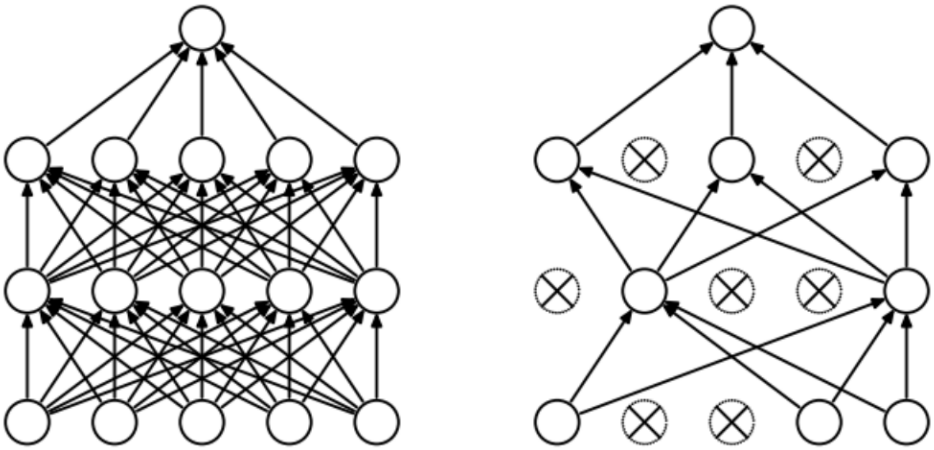


Figure 3.8: Dropout in neural network

Dropout is a regularization technique for neural network models proposed by Srivastava in [118], as shown in Fig. 3.8. Dropout is a technique where randomly selected neurons are ignored during training. They are dropped-out randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

In statistics, overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably" [83]. An overfitted model is a statistical model that contains more parameters than can be justified by the data [31]. The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e. the noise) as if that variation represented underlying model structure. Thus, overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. There are quite a number of ways in which we can prevent our model from overfitting, such as dropout and use of cross validation, which we discuss in Chapter 4.

The reason why dropout is able to prevent overfitting is that it decreases the sensibility of the output to the input. As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations. The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data [65].

TensorFlow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. In this project, we developed a machine learning computer code by using TensorFlow.

Created by Google and written in C++ and Python, TensorFlow is perceived to be one of the best open source libraries for numerical computation. TensorFlow is good for advanced projects, such as creating multilayer neural networks. Its used in voice/image recognition and text-based apps (like Google Translate). There are three main advantages of TensorFlow:

1. Comes with Tensorboard for Network Visualization: This is probably the main reason why TensorFlow is attracting developers towards it. Easy visualization of even complex neural networks.

2. Ever Evolving: TensorFlow is relatively new but is evolving fast. All the algorithms are written by experts. New versions of algorithms are updated. A lot of big companies use it in its projects, so we can be pretty sure that they are amongst the best versions of algorithms that one can find.

3. Creating the whole network with placeholders: Its backed by a large community of devs and tech companies. Placeholders enable us to separate the network configuration phase and the evaluation phase. Although it may seem as an overhead, it actually helps to scale up the network without any modifications being made.

4. Other factors: such as providing model serving, supporting distributed training and providing a lot of documentation and guidelines.

Implementation

There are four layers in the developed DNN code, with the learning rate of 0.0035, and the dropout of 0.05. We use the ReLU as the activation function, and we ran it in 18000 steps. In general, the number of hidden neurons should be between the size of the input layer and the size of the output layer. The input parameters of neural network are the plastic displacements or the permanent displacements along both the horizontal and vertical directions of all the nodes of the FEM mesh, except the boundary nodes, of the cantilever beam. The size of input layer is 60 or more after the numerical process, depending on the mesh of the model. The size of output layer is 3 or 8, depending on different numerical models. Thus, we choose from eight to sixty hidden layers in our machine learning code. After trying different numbers of hidden layers, we found that four hidden layers neural network provides good computation results for the FEM mesh size used and the numerical model. More hidden layers will lead to much more calculation time, but not much better results. Thus, we choose four hidden layers as the default structure for our machine

learning test code. The number of hidden neurons should be less than twice the size of the input layer. Hence, we choose the first hidden layer 32 neurons, the second and third hidden layers have 64 neurons each. The final hidden layer has 8 neurons.

3.3 Methodology for Elasto-plastic Shell Structures

We have developed an artificial intelligence based machine learning, in fact a deep learning method, to determine and identify loading conditions for structures and materials based on the permanent plastic deformation of the damage state, which provides an effective solution to the failure analysis load identification problem. Using machine learning methods, one may make relatively accurate predictions on the problems that were difficult to be solved otherwise. In the research, we choose permanent plastic deformation as the measurable structural responses that are fed into a deep learning neural network as input, with the predicted loading conditions as output. This procedure is illustrated in Fig. 3.9.

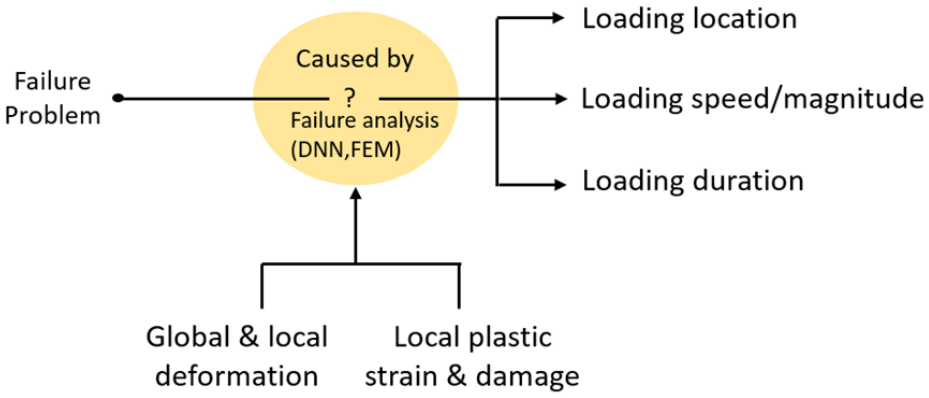


Figure 3.9: Illustration of machine learning approach to identify structure failure load conditions.

In the rest of the section, we first begin with a detailed description of the machine learning algorithm used. Then we outline the process for gathering the required data to train the machine learning algorithms before building up the deep neural network. This is followed by examples that demonstrate how to use proposed deep-learning neural network to solve the inverse problem. In particular, we illustrate the solution process for a hemispherical steel shell of inelastic materials with static and dynamic impact loadings at different locations. We seek to demonstrate through these examples that the deep learning algorithms developed in this work can accurately identify both static loading and impact loading conditions based on observed permanent plastic strain or deformation.

Identification Method

The inverse problem that we are solving in this study is a collision between a hemispherical shell and a rigid cylindrical body. The shell is fixed on the ground while a rigid cylindrical body randomly impacts on it with different velocities. The loading parameters needed to predict are (1) the first contact point on the shell, (2) the normal and tangential velocities of the rigid body, and (3) the impact duration. The observed or measured data are permanent plastic deformation of the hemispherical shell. The first contact point is worth to be predicted because it reflects the final positions of two objects before a collision. It is different from the maximum deformation points because the rigid body has velocities on normal and tangential directions. It would slide on the shell surface after contact. Thus, we cannot figure out the first contact point through the deformation visibly. Once getting these three parameters, we can fully recover the impact process through simulation.

To improve prediction correctness among a huge dataset which consists of different kinds of data, Gao and Mosalam [41] proposed a Structural ImageNet, which is a hierarchy tree, to solve a complex classification problem by dividing it into several binary classification problems. Similarly, the inverse problem of collision is also complex because there is no linear relationship between the plastic deformation and the loads. Meanwhile, each impact has different velocities, duration, and locations. The relationships between the deformation and each of these parameters are very different. It is difficult to regress all of these parameters in a single neural network. Inspired by the idea of Structural ImageNet, we propose a deep learning identification method to identify the impact loads as shown in Fig. 3.10.

Firstly, we trained two DNNs with the whole database to enable them to recognize the first contact point and the duration of the impact, respectively. It should be noted that, for a 3D structure, it needs a great number of measurement points on its surface to measure the deformation precisely. In this study, the 2m-diameter shell structure has more than 6400 measurement points and each point has three directions of deformation which result in $18k+$ features in each set of data. For a simplified model, the measurement points would be more than $70k$. Thus, we have to pre-process the data before putting them into the neural network. In this research, we adopted principal component analysis to reduce the dimensions of our data to make the training process more efficient. After these two DNNs were trained well, we used them to identify the first contact point and the impact duration of the identification object.

Secondly, with the identified location and duration, we extracted the training data in the neighborhood region of the predicted location from the whole database. For example, if we identify the first contact location is at the point (x_1, y_1), then we can extract the training data from a small region centered at (x_1, y_1), and use the extracted data to train the third DNN to make it specific to velocity identification. Finally, putting the new plastic deformation to this DNN, the velocity of the impact would be obtained. When all those three parameters were obtained, we put them into the FEM model to recover the impact process and get the final permanent plastic deformation. Com-

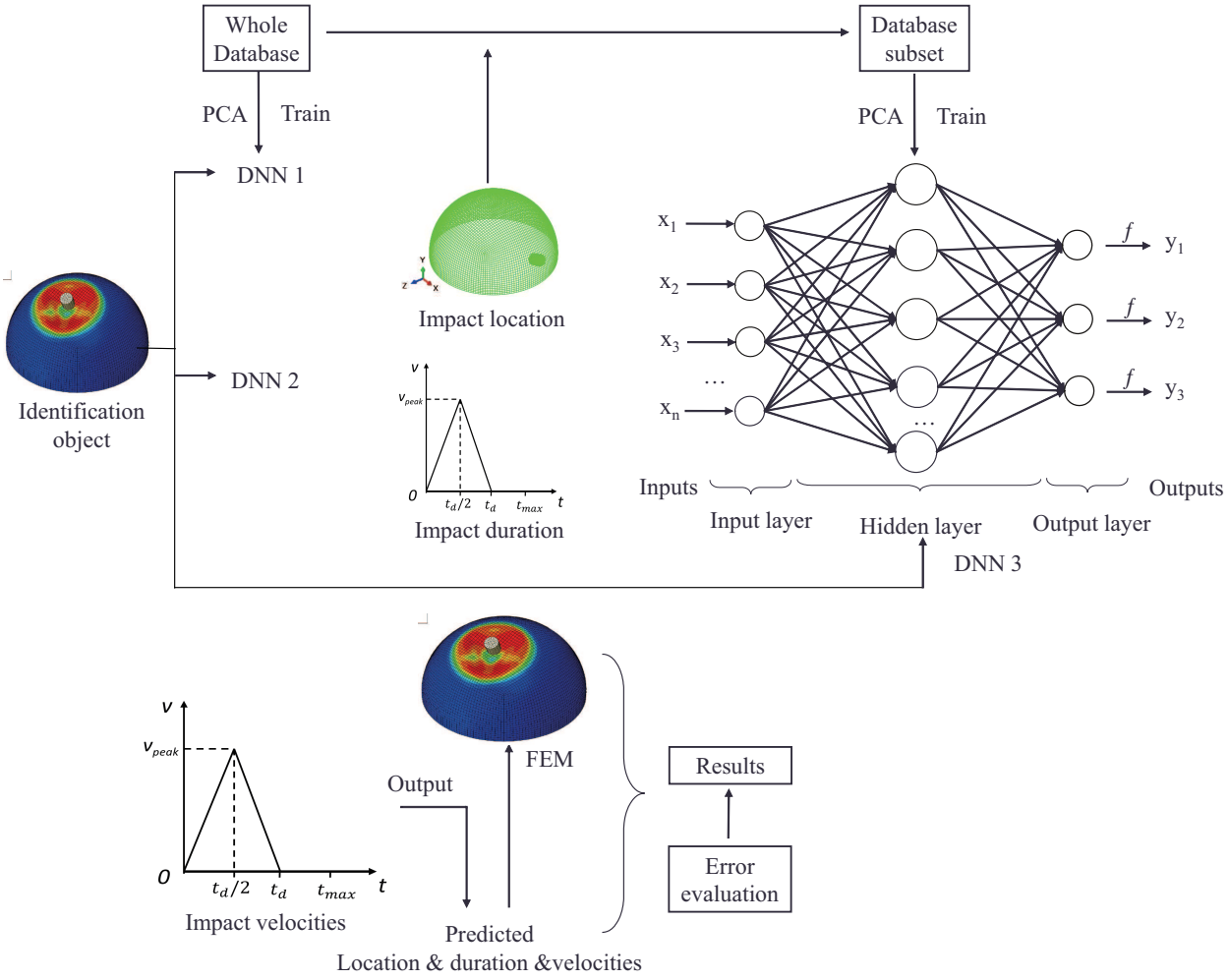


Figure 3.10: Impact load identification method

paring the simulated deformation with the ground truth deformation, if the error between them is less than requirement, the impact loads are found.

In summary, the computation flowchart of the entire modeling and computation process is shown in Fig. 3.11. The main stages are as follows:

(1) We first build a neural network and use TensorFlow to predict location and duration based on all the training data.

(2) Based on the predicted location and duration, we can obtain the new sets of training data from data filtering or decomposition, and then we build another new neural network.

(3) We use the new neural network to predict the magnitude of the contact velocity.

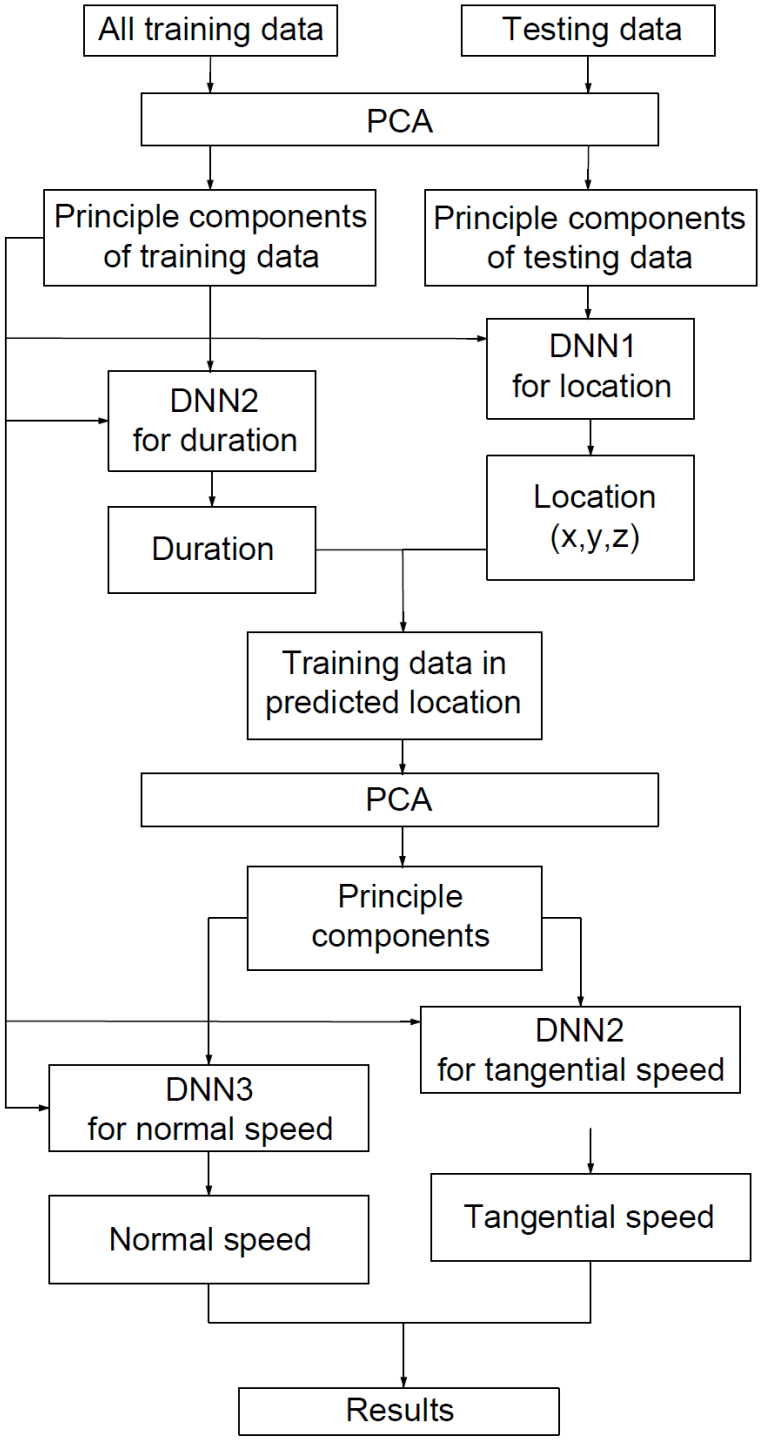


Figure 3.11: Flowchart of data processing

Principal Component Analysis

In this work, we adopted a basic model of artificial neural network as our algorithm. As it is mentioned before, in order to reduce dimension, we also introduce a dimension reduction technique — principle component analysis (PCA). Principle components are a sequence of projection of data, mutually uncorrelated and ordered in variance [52].

Principal component analysis is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. If there are n observations with p variables, then the number of distinct principal components is min(n-1, p). This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing n observations) are an uncorrelated orthogonal basis set.

PCA is sensitive to the relative scaling of the original variables. PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualized as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced [99, 57].

Even though there are some cons of PCA, such as a high computational cost, therefore it cannot be applied to very large datasets, and not good when working with fine-grained classes, the pros of PCA, such as reduction in size of data, allowing estimation of probabilities in high-dimensional data, and being able to render a set of components that are uncorrelated, are very obvious in our project. Thus, we use PCA method in our machine learning algorithm.

By using the dimension reduction technique, we first subtract the mean of all rows of data, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, so that we can find directions of high variance in data field. For a $n \times p$ matrix $X \in \mathbb{R}^{n\times p}$, one can do singular value decomposition (SVD) to decompose the X matrix into three parts as follows:

$$X = USV^T \tag{3.5}$$

where $U \in \mathbb{R}^{n\times n}$, which is an orthogonal matrix, $S \in \mathbb{R}^{n\times p}$, which is a diagonal matrix, and $V \in \mathbb{R}^{p\times p}$, which is also an orthogonal matrix.

By using the above techniques, we can then derive the principle components from US, and reduce the dimension of data in to a relatively small one that we desired. After the data transfor-

mation, we did the analysis in order to choose the optimal dimension of the data set for training. In our case we choose 96% of data and reduce the large data set into 10 columns. These 10 columns represent for around 96.038% of the original data. After dimension reduction, the data that are sent into our deep neural network model is the input data.

The machine learning models of this simulation is the same with those used in the simulation of cantilever beams of inelastic materials.

Adam Optimization Algorithm

The choice of optimization algorithm for deep learning model can mean the difference between good results in minutes, hours, and days. The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing.

Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in 2015 [71, 71]. Adam is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. When introducing the algorithm, the authors list the attractive benefits of using Adam on non-convex optimization problems, as follows: straightforward to implement, computationally efficient, little memory requirements, invariant to diagonal rescale of the gradients, well suited for problems that are large in terms of data and/or parameters, appropriate for non-stationary objectives, appropriate for problems with very noisy/or sparse gradients and hyper-parameters have intuitive interpretation and typically require little tuning.

Selection of batch size

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameter, which refers to the number of training examples utilized in one iteration in machine learning [84]. The batch size can be one of three options:

Batch mode: where the batch size is equal to the total dataset thus making the iteration and epoch values equivalent. When all training samples are used to create one batch, the learning algorithm is called batch gradient descent or full batch learning.

Mini-batch mode: where the batch size is greater than one but less than the total dataset size. Usually, a number that can be divided into the total dataset size.

Stochastic mode: where the batch size is equal to one. Therefore the gradient and the neural network parameters are updated after each sample.

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches [60, 7].

The advantage of full batch learning is that gradient descent tends to be more accurate since the total dataset could represent all the data better than part of the dataset. Also, the number of epoches tends to be smaller.

Implementation

Table 3.1: Hyper-parameters of the DNN

Architecture / hyper-parameters	Values
Learning rate	0.0025
Learning rate decay rate	0.99
Dropout rate	0.1 / 0.15
Training steps	8000
Loss function	MSE
Activation function	Relu
Batch size	Full
Optimizer	Adam

We developed a machine learning computer code by using TensorFlow. Fig. 3.12 shows the illustration of structure of neural network. The hyper-parameters of our DNN model are listed in Table 5.17. The implementation of the model was coded by python language.

3.4 Methodology for Car Collisions

Identification Method

The main methods we use in car collision is the same with the methods we use in elasto-plastic shell structures. However, in the project, we implement three deep neural networks and the process is:

(1) Building a neural network and use TensorFlow to predict the offset between two cars based on all the training data.

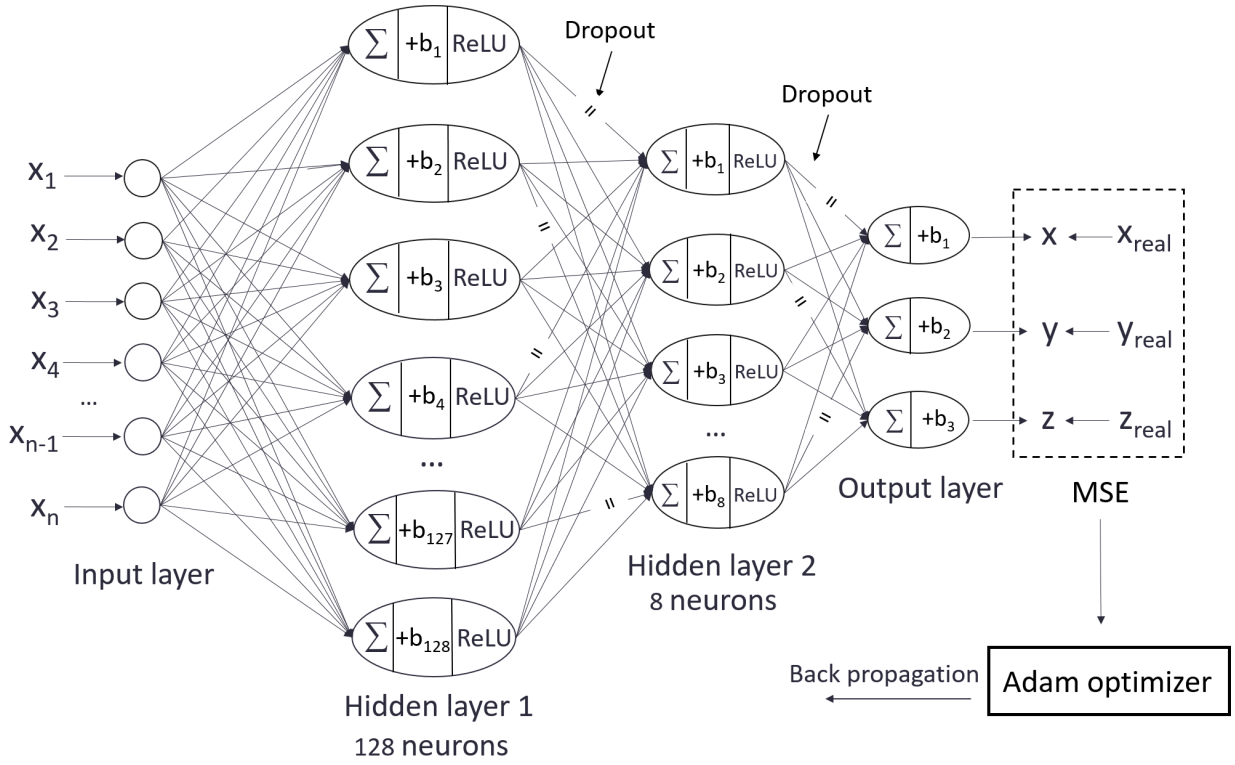


Figure 3.12: Illustration of structure of the neural network

(2) Based on the predicted offset, obtaining the new sets of training data from the whole training data using data filtering, and then building the second neural network.

(3) Using the second neural network to predict the magnitude of the velocities of two cars.

(4) Based on the predicted offset and velocities, using data filtering again to obtain the new sets of training data from the filtered training data in step (2), and then building the third neural network.

(5) Using the second neural network to predict the magnitude of the velocities of two cars.

The technology we use in this study is data filtering. Data filtering is the process of choosing a smaller part of data set and using that subset for viewing or analysis. Filtering is generally (but not always) temporary the complete data set is kept, but only part of it is used for the calculation [50]. Data filtering in IT can refer to a wide range of strategies or solutions for refining data sets. This means the data sets are refined into simply what a user (or set of users) needs, without including other data that can be repetitive, irrelevant or even sensitive. Different types of data filters can be used to amend reports, query results, or other kinds of information results.

Typically, data filtering will involve taking out information that is useless to a reader or information that can be confusing. Generated reports and query results from database tools often result in large and complex data sets. Redundant or impartial pieces of data can confuse or disorient a user. Filtering data can also make results more efficient.

To bring in these three neural networks, the accuracy and speed gets improved a lot, which is discussed in detail in Chapter 4 and Chapter 6.

Data Standardization

Machine learning algorithms make assumptions about the dataset we are modeling. Often, raw data is comprised of attributes with varying scales. Although not required, we can get a boost in performance by carefully choosing methods to rescale data, which is data standardization.

The result of standardization (or Z-score normalization) is that the features will be rescaled so that theyll have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$, where μ is the mean (average) and σ is the standard deviation from the mean. Standard scores (also called z scores) of the samples are calculated as follows:

$$z = \frac{x - \mu}{\sigma} \tag{3.6}$$

Standardizing the features so that they are centered around 0 with a standard deviation of 1 is not only important if we are comparing measurements that have different units, but it is also a general requirement for many machine learning algorithms, such as PCA and neural network [96, 46]. Data standardization forces several different features into same range and standard deviation so that they are comparable regardless of possible different ranges of features and the influence of different units. In our study, different features have several different ranges, such as [0, 10^{-6}] and [0, 1]. That's why standardization is used in this study.

Standardization is important in PCA since PCA is a variance maximizing exercise. It projects original data onto directions which maximize the variance. It gives more emphasis to those variables having higher variances than to those variables with very low variances while identifying the right principle component. In order to decrease the loss-information influence of PCA, standardization should be done before that.

Implementation

In our neural network model, we use TensorFlow to perform analysis and simulation of neural network. There is also one significant limitation of TensorFlow: the only fully supported language is Python. In our Python code, we have three hidden layers and the sizes are $256, 64, 8$. The learning rate is 0.0035 and the number of steps is $50,000$. The dropout parameter is 0.05 to prevent

overfitting. The principle component analysis parameter is 5 and the ratio is 99.9896.

Several theoretical contributions to machine learning algorithms in the car collision study is highlighted in Chapter 4.

Chapter 4

Theoretical Contribution to Algorithms

This chapter highlights several theoretical contributions to machine learning algorithms in this study, especially in car collision simulations.

4.1 Activation Function

In artificial neural networks, the activation function of a node defines the output of that node, or "neuron," given an input or set of inputs. This output is then used as input for the next node and so on until a desired solution to the original problem is found.

It maps the resulting values into the desired range such as between 0 to 1 or -1 to 1 etc. (depending upon the choice of activation function). For example, the use of the logistic activation function would map all inputs in the real number domain into the range of 0 to 1 [116, 59, 117].

A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, only nonlinear activation functions allow such networks to compute nontrivial problems using only a small number of nodes [75]. In artificial neural networks, this function is also called the transfer function.

In the car collision study herein, we use four different activation functions: ReLU, sigmoid, tanh and softmax.

ReLU (Rectified Linear Unit)

$$f(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases} \tag{4.1}$$

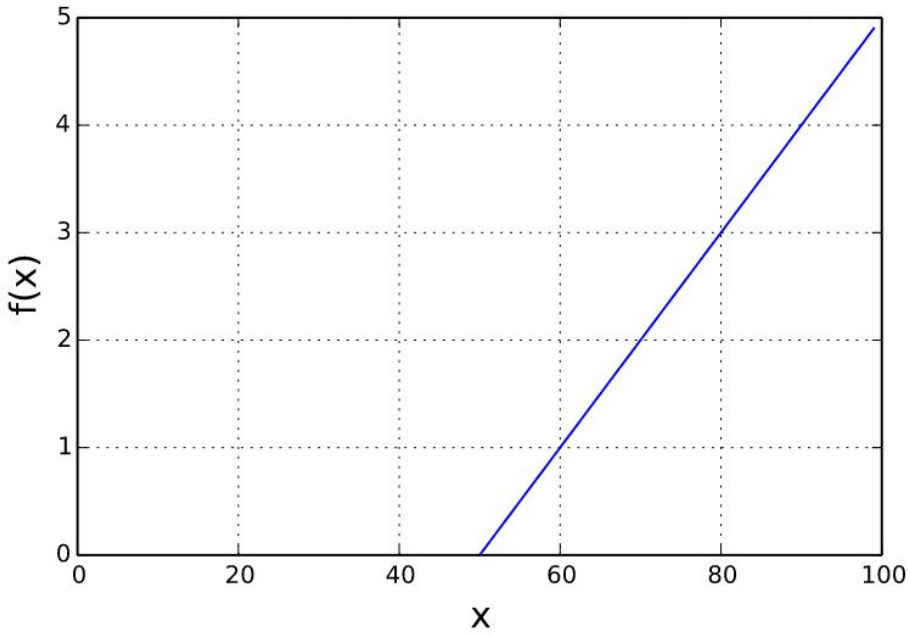


Figure 4.1: Relu function plot

A rectified linear unit has the output 0 if its input is less than or equal to 0, otherwise, its output is equal to its input. It is also more biologically accurate. This has been widely used in convolutional neural networks. It is also superior to the sigmoid and tanh activation function, as it does not suffer from the vanishing gradient problem. Thus, it allows for faster and effective training of deep neural architectures.

However, being non-differentiable at 0, ReLU neurons have a tendency to become inactive for all inputs i.e. they die out. This can be caused by high learning rates, and can thus reduce the models learning capacity. This is commonly referred to as the Dying ReLU problem [16].

The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning.

The function and its derivative both are monotonic. But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

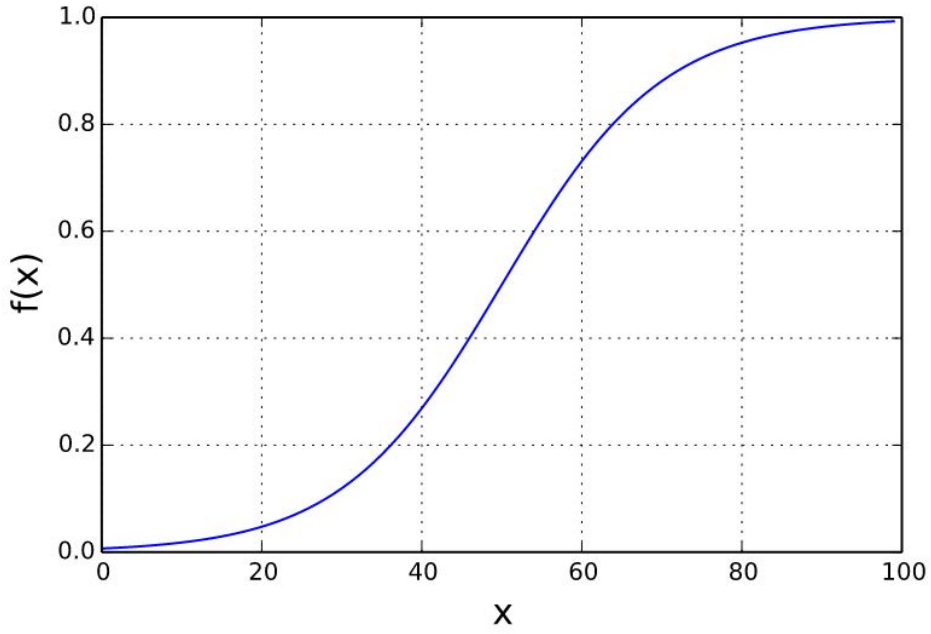


Figure 4.2: Sigmoid function plot

Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

The sigmoid or logistic activation function maps the input values in the range (0,1), which is essentially their probability of belonging to a class. So, it is mostly used for multi-class classification.

The Sigmoid Function curve looks like a S-shape. The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice [89].

The function is differentiable. That means, the slope of the sigmoid curve at any two points can be computed. The function is monotonic but functions derivative is not. The logistic sigmoid function can cause a neural network to get stuck at the training time. The softmax function is a more generalized logistic activation function which is used for multiclass classification.

However, like tanh, it also suffers from the vanishing gradient problem. Also, the output it produces is not zero-centered, which causes difficulties during optimization. It also has a low convergence rate.

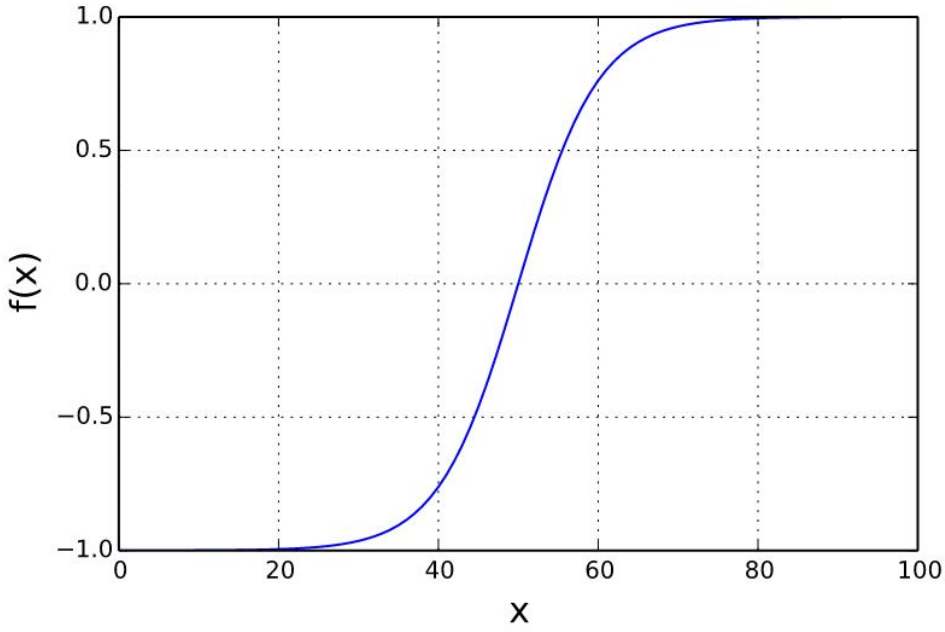


Figure 4.3: Tanh function plot

Tanh

$$f(x) = tanh(x) \tag{4.3}$$

The tanh non-linearity compresses the input in the range (-1,1). It provides an output which is zero-centered. So, large negative values are mapped to negative outputs, similarly, zero-valued inputs are mapped to near zero outputs. Also, the gradients for tanh are steeper than sigmoid, but it suffers from the vanishing gradient problem [66].

The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

The function is differentiable. The function is monotonic while its derivative is not monotonic. The tanh function is mainly used classification between two classes.

Both tanh and logistic sigmoid activation functions are used in feed-forward nets.

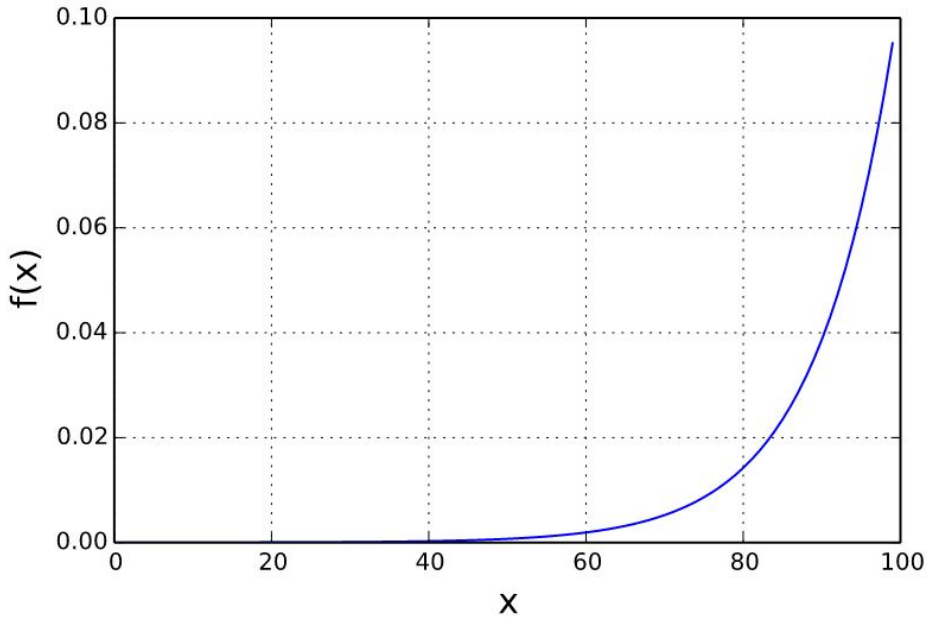


Figure 4.4: Softmax function plot

Softmax

$$f(x) = \frac{e^x}{\sum_{i=0}^{k} e^{x_i}} \tag{4.4}$$

The softmax functions output tells us the probabilities that any of the classes are true, so it produces values in the range (0,1). It highlights the largest values and tries to suppress values which are below the maximum value, its resulting values always sum to 1. This function is widely used in multiple classification logistic regression models [28].

Softmax function calculates the probability distribution of the event over n different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities will be helpful for determining the target class for the given inputs.

The main advantage of using Softmax is the output probabilities range. The range is 0 to 1, and the sum of all the probabilities is equal to one. If the softmax function is used for multi-classification model, it returns the probabilities of each class and the target class will have the high probability.

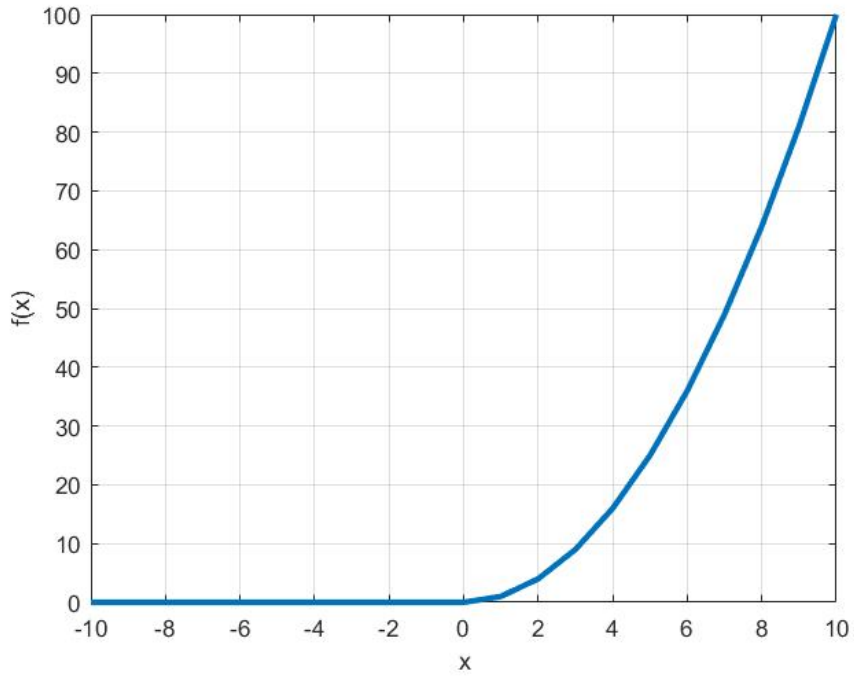


Figure 4.5: Relu Square function plot

The formulation computes the exponential (e-power) of the given input value and the sum of exponential values of all the values in the inputs. Then the ratio of the exponential of the input value and the sum of exponential values is the output of the softmax function.

In summary, softmax has two properties: the calculated probabilities are in the range of 0 to 1; the sum of all the probabilities is equals to 1.

ReLU Square

$$f(x) = \begin{cases} 0, x < 0 \\ x^2, x \geq 0 \end{cases} \tag{4.5}$$

Since the ReLU is the most used activation function in the world right now, we develop a modified method, ReLU Square method, based on ReLU. A rectified linear unit has the output 0 if its input is less than or equal to 0, otherwise, its output is equal to its square of input.

The three major benefits of ReLU Square are sparsity, a reduced likelihood of vanishing gradient and a larger nonlinearity.

One major benefit is the reduced likelihood of the gradient to vanish. This arises when x ¿ 0. In this regime the gradient has a linear value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases. The constant gradient of ReLU Square results in faster learning.

The other benefit of ReLUs is sparsity. Sparsity arises when $x \leq 0$, even larger than ReLU. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations.

The third benefit is a larger non-linearity. The purpose of the activation function is to introduce non-linearity into the network. In turn, this allows to model a response variable (aka target variable or class label) that varies non-linearly with its explanatory variables. Here non-linear means that the output cannot be reproduced from a linear combination of the inputs. Without a non-linear activation function in the network, a neural network, no matter how many layers it had, it would behave just like a single-layer prediction, because summing these layers would give just another linear function. ReLU Square brings a larger non-linearity to neural network, comparing to ReLU.

However, it has the same limitation with ReLU method, which is, being non-differentiable at 0, ReLU Square neurons have a tendency to become inactive for all inputs i.e. they die out. This can be caused by high learning rates, and can thus reduce the models learning capacity. All the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly.

4.2 Cost Function

All the algorithms in machine learning rely on minimizing or maximizing a function, which is called the objective function. The group of functions that are minimized are called loss functions. A loss function is a measure of how good a prediction model does in terms of being able to predict the expected outcome.

There is not a single loss function that works for all kind of data. It depends on a number of factors including the presence of outliers, choice of machine learning algorithm, time efficiency of gradient descent, ease of finding the derivatives and confidence of predictions. In our study, we use mean square error as the loss function [98, 44].

Cost function, as known as risk function or objective function, is what we want to minimize. In ML, cost functions are used to estimate how badly models are performing. Put simply, a cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between x and y. This is typically expressed as a difference or distance between the predicted value and the actual value. The cost function, also referred to as loss or error, can be estimated

by iteratively running the model to compare estimated predictions against ground truth, the known values of y.

The objective of a ML model, therefore, is to find parameters, weights or a structure that minimizes the cost function.

In the car collision study herein, we use two different cost functions: Mean Loss and Maximum Loss.

Mean Loss

$$J(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(X_i), y_i)) \tag{4.6}$$

Maximum Loss

$$J(h) = max_{i=1}^{n} L(h(X_i), y_i)) \tag{4.7}$$

4.3 Feature Selection

In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for four reasons: simplification of models to make them easier to interpret by researchers/users; shorter training times; to avoid the curse of dimensionality and enhanced generalization by reducing overfitting [61, 9].

The central premise when using a feature selection technique is that the data contains some features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information [9]. Redundant and irrelevant are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated [48].

Feature selection techniques should be distinguished from feature extraction. Feature extraction creates new features from functions of the original features, whereas feature selection returns a subset of the features. Feature selection techniques are often used in domains where there are many features and comparatively few samples (or data points). Archetypal cases for the application of feature selection include the analysis of written texts and DNA microarray data, where there are

many thousands of features, and a few tens to hundreds of samples.

In this study, plastic strain, residual deformation and residual displacement are all chosen to be features separately and machine learning simulations are performed. Based on the error, residual displacement is the best feature. It is because that it has the largest variance among the data of the three features.

4.4 Data Filtering

As mentioned in Chapter 3, we implement two deep neural networks. Between the three deep neural networks, data filtering is implemented to get rid of the useless data.

In a large size database, normally there will be incomplete data. The incomplete data may be outlier values or error data. Hence, data filtering will be needed for the preprocessing of data. Data filtering is the process of getting rid of noise such as outlier values and error data from raw data to make the data clean and be proper for further processing [126]. There are many techniques of data filtering such as moving average filtering [5], local regression filtering [97], Savitzky-Golay filtering [17, 121, 85], and Hamming window filtering [127]. In this study, we filter data based on the result of the previous neural network. There are four benefits of that:

1. Culling. A good filtering tool enables to cull the data set down to a more manageable volume so that useful data reports can be produced and shared with others.

2. Save and Rerun. The software products allows to save the filters we create so we can rerun them at a later time when new information is added to the case.

3. Isolate. Pre-search filtering can isolate a subset of records based on fields such as status, evaluation and linked issues, creating slices of data for subsequent processing.

4. Small Production. An effective data set filter allows to narrow down to a smaller set of records for light document production.

4.5 Metrics to Evaluate Machine Learning Algorithms

Cross-validation

In this study, we use cross-validation to evaluate the machine learning model.

Cross-validation, sometimes called rotation estimation, or out-of-sample testing is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction,

Figure 4.6: Overview of k-fold cross-validation method [22]

and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set).The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem) [74, 123, 15].

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, in most methods multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to give an estimate of the model's predictive performance [51, 110].

There are several functions of cross-validation method. First, it helps us evaluate the quality of the model. Also, it helps us select the model which will perform best on unseen data and avoid overfitting and underfitting.

In summary, cross-validation combines (averages) measures of fitness in prediction to derive a more accurate estimate of model prediction performance. One of the popular cross validation

examples is k-fold validation method.

The idea behind cross-validation is to estimate the model predictive performance on unseen data. Cross-validation does this by repeating the experiment multiple times, using all the different parts of the training set as validation sets. This gives a more accurate indication of how well the model generalizes to unseen data.

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples, as shown in Fig. 4.6. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k - 1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general k remains an unfixed parameter [74, 130, 39]. In addition, k-fold cross-validation estimator has a lower variance than a single hold-out set estimator, which can be very important if the amount of data available is limited. In this study, we use 10-fold cross-validation method to judge if a machine learning model is good or not.

L^2 Norm

Despite cross-validation method, a new metrics to evaluate machine learning algorithms is developed, which is L^2 norm.

Before using L^2 norm, relative error (RE) is a common way to estimate the final error, such as the error of predicted velocity or hitting position and the true vlocity/hitting position. It is used as a measure of precision and is the ratio of the absolute error of a measurement to the measurement being taken. Relative error gives an indication of how good a measurement is relative to the size of the quality being measured. In other words, this type of error is relative to the size of the item being measured [101]. RE is expressed as a percentage and has no units.

In linear algebra, functional analysis, and related areas of mathematics, a norm is a function that assigns a strictly positive length or size to each vector in a vector space except for the zero vector, which is assigned a length of zero. A seminorm, on the other hand, is allowed to assign zero length to some non-zero vectors (in addition to the zero vector). A norm must also satisfy certain properties pertaining to scalability and additivity which are given in the formal definition below [95, 102].

A simple example is two dimensional Euclidean space $R2$ equipped with the "Euclidean norm", or "L^2 norm". Elements in this vector space are usually drawn as arrows in a 2-dimensional cartesian coordinate system starting at the origin (0, 0). The Euclidean norm assigns to each vector the length of its arrow. Because of this, the Euclidean norm is often known as the magnitude [129].

A vector space on which a norm is defined is called a normed vector space. Similarly, a vector space with a seminorm is called a seminormed vector space. It is often possible to supply a norm for a given vector space in more than one way.

The L^2 norm $|x|$ is a vector norm defined for a complex vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \tag{4.8}$$

by

$$|x| = \sqrt{\sum_{k=1}^{n} |x_k|^2} \tag{4.9}$$

where $|x_k|$ on the right denotes the complex modulus. The L^2 norm is the vector norm that is commonly encountered in vector algebra and vector operations (such as the dot product), where it is commonly denoted $|x|$. However, if desired, a more explicit (but more cumbersome) notation $|x|_2$ can be used to emphasize the distinction between the vector norm $|x|$ and complex modulus together with the fact that the L^2 norm is just one of several possible types of norms.

In this study, for the difference matrix between the original and the predicted displacement value, L^2 norm is calculated. L^2 norm represents the difference between the original displacement and the predicted displacement. By this way, we can directly tell the accuracy of the current machine learning model. We use

$$d1 = \begin{bmatrix} d1_1 \\ d1_2 \\ . \\ . \\ . \\ d1_n \end{bmatrix} \tag{4.10}$$

to represent the original residual displacement and

$$d2 = \begin{bmatrix} d2_1 \\ d2_2 \\ . \\ . \\ . \\ d2_n \end{bmatrix} \tag{4.11}$$

the represent the predicted residual displacement. The matrix is denoted by

$$d = \begin{bmatrix} d1_1 - d2_1 \\ d1_2 - d2_2 \\ \cdot \\ \cdot \\ \cdot \\ d1_n - d2_n \end{bmatrix} \tag{4.12}$$

The L^2 norm of the matrix is represented by

$$L^2 = \sqrt{\sum_{k=1}^{n} |d1_k - d2_k|^2} \tag{4.13}$$

To get rid of the influence of units, we use the modified L^2 norm:

$$L^2_{modified} = \sqrt{\frac{\sum_{k=1}^{n} |d1_k - d2_k|^2}{A}} \tag{4.14}$$

in which, A represents the surface area of car model.

Chapter 5

Simulation and Results of Cantilever Beams with Inelastic Materials and Elasto-Plastic Shell Structures

Based on the methods described in Chapters 3 and 4, this chapter developed three simulations of novel machine (deep) learning computational framework to determine and identify damage loading parameters (conditions) for structures and materials, including cantilever beams with inelastic materials and elasto-plastic shell structures, based on the residual plastic deformation distribution or residual displacement.

5.1 Cantilever Beams with Inelastic Materials

Data Collection

Geometric and material properties

A 2D cantilever beam was chosen for the study, which has a length of 5 m and a width of 1 m. The finite element model of the cantilever beam was developed with the ABAQUS software. Plane strain condition was assumed throughout this study, and CPE4R element was used, which is a 4-node bilinear plane strain quadrilateral element [113]. In order to ensure the accuracy of the numerical calculation, different mesh sizes of the beam were chosen for the convergence analysis, and the final mesh size was chosen as 0.25m. Accordingly, the number of nodes is 105, and the number of elements is 80.

We choose AISI 4340 steel (33 HRc) as the material of the cantilever beam, which is modeled by using the Johnson-Cook plasticity model (see [47, 63]). It is model as a thermo-elastoplastic solid, as expressed in the following equations,

$$\sigma = [A + B(\epsilon^p)^n][1 + Cln(\frac{\dot{\epsilon}^p}{\dot{\epsilon}_0}][1 - (T^*)^m] \tag{5.1}$$

Table 5.1: Mechanical properties of AISI 4340 steel (33 HRc) (From [47, 63]).

Density (kg/m^3)	7830
Young's modulus (GPa)	208
Poisson ratio	0.3
$A(MPa)$	792
$B(MPa)$	510
C	0.014
n	1.03
m	0.26
D_1	0.05
D_2	3.44
D_3	-2.12
D_4	0.002
D_5	0.61
$\dot{\epsilon}_0(s^{-1})$	1

$$T^* = \frac{T - T_0}{T - T_m} \tag{5.2}$$

where σ is the flow stress, ϵ^p is the equivalent plastic strain, $\dot{\epsilon}^p$ is the strain rate, $\dot{\epsilon}_0$ is the reference strain rate, A, B, C, m, n are material constants, and T^* is the homologous temperature which is related to the absolute temperature T, the reference temperature T_0 and the melting temperature T_m.

The critical failure strain is defined as [47, 63]:

$$\epsilon_f = [D_1 + D_2 e^{(D_3 \sigma^*)}][1 + D_4 ln(\frac{\dot{\epsilon}^p}{\dot{\epsilon}_0})][1 + D_5 T^*] \tag{5.3}$$

where D_i are material constants, and σ^* is the dimensionless pressure-stress ratio.

Material parameters of AISI 4340 steel are listed in Table 5.1.

Boundary conditions and loads

The finite element model of the cantilever beam is presented in Fig.5.1. All degrees of freedom of the four nodes on the left at $x = 0$ m are rigidly fixed. As shown in this figure, seven numbered nodal points were chosen to apply loads and corresponding sets of simulation data were generated. In most cases when loads are applied to the three points closer to the support of the cantilever beam (nodes 5, 6 and 7 in Fig.5.1), the residual displacement and plastic strain are all zero at all the nodes, which leads to the issue multiple-answer issue to DNN since zero residual displacement

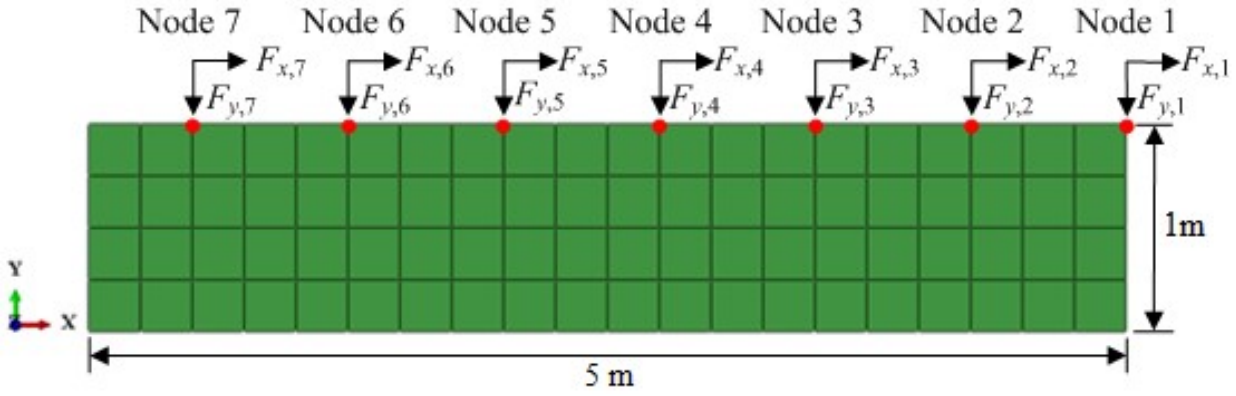


Figure 5.1: Finite element model of the cantilever beam and the loading positions

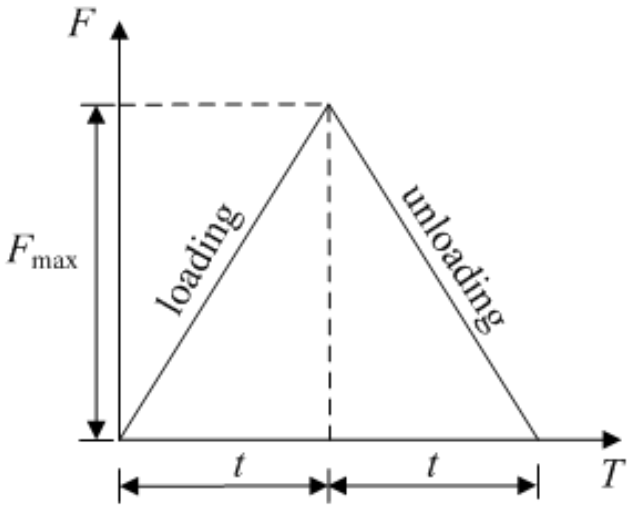


Figure 5.2: Dynamic loading time history

corresponds to three different locations. Thus, the loading at these points are excluded and only the loading points 1-4 are considered in the training, as shown in Fig.5.1.

Both static and dynamic responses of the cantilever beam under concentrated loading forces are computed by using ABAQUS. It should be noted that the dynamic loading history of each concentrated force acting on the beam, which can cause plastic deformation of the beam, is featured with a bi-linear loading and unloading curve of bandwidth t and loading amplitude F_{max}, as shown in Fig.5.2.

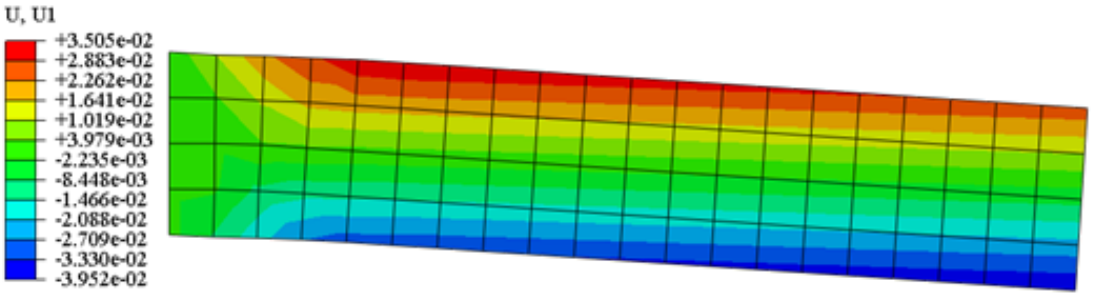
Four case studies were considered to establish the database and verify the deep learning algorithm:

Table 5.2: Loads for the static analysis

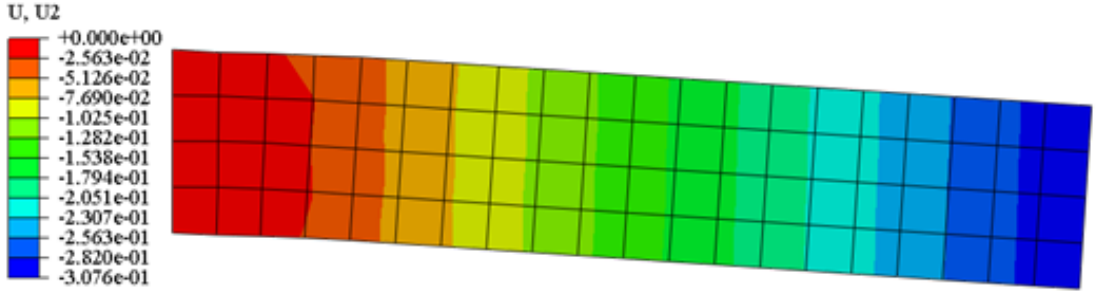
Node	F_{max} (N)	t
1	$5.0 \times 10^7 - 8.0 \times 10^7$	0.5
2	$6.0 \times 10^7 - 9.0 \times 10^7$	0.5
3	$8.0 \times 10^7 - 12.0 \times 10^7$	0.5
4	$8.0 \times 10^7 - 14.0 \times 10^7$	0.5

Table 5.3: Loads for the dynamic analysis

Node	F_{max} (N)	t (s)
1	$7.1 \times 10^7 - 7.3 \times 10^7$	0.01-0.03
2	$8.1 \times 10^7 - 9.3 \times 10^7$	0.01-0.03
3	$10.0 \times 10^7 - 11.8 \times 10^7$	0.01-0.03
4	$12.0 \times 10^7 - 14.8 \times 10^7$	0.01-0.03



(a)



(b)

Figure 5.3: Lastic displacement distribution: (a) Plastic displacement along horizontal direction, and (b) Plastic displacement along vertical direction.

(1) Prediction of static loads acting on the numbered nodes (also referred as training nodes in this chapter). Static loads simultaneously were imposed to one up to four numbered nodes shown in Fig.5.1. Responses of the beam under static loads of different amplitudes were numerically predicted, and the database for the deep learning was developed. Then, a different deformation of the beam caused by loads applied to the numbered nodes was provided by ABAQUS program, and the amplitude of the static load was predicted by the deep learning algorithm and compared with the exact solution.

(2) Prediction of static loads acting between the numbered nodes. In this case study, the database for the deep learning was also developed by applying static forces to the numbered nodes. Then, a deformation of the beam caused by a load acting on a node between the two adjacent numbered nodes (as shown in Fig.5.1) was given by ABAQUS program. Amplitude and position of the static load were predicted by the deep learning algorithm.

(3) Prediction of the impact nodal load acting on the numbered nodes. Impact loads imposed to one numbered node. Responses of the beam under impact load with different amplitudes and durations were numerically predicted. Then, a different deformation of the beam caused by an impact load acting on this numbered node was given, and both the amplitude and duration of the impact load were predicted.

(4) Prediction of the impact load acting between the numbered nodes. This case study is similar to the second case study. Position, amplitude and duration of the impact load were all predicted by the deep learning algorithm.

The bi-linear pulse load–time history curve is assumed throughout this study as shown in Fig.5.2 for dynamic loading. For the static analysis, the step time T equals 1, and durations of loading and unloading are 0.5. For the dynamic response, duration of loading and unloading are significantly reduced to orders of 0.01-0.03s to simulate the impact loads. Meanwhile, in order to minimize the influence of inertial effects and get the final stable deformation of the beam, the step time T for the dynamic analysis is set as 50-100 s.

For the static and dynamic analysis, amplitude, duration and step time for the first two case studies are listed in Table 5.2 and Table 5.3, respectively. For the multi-points condition, amplitude of the load is reduced. For static analysis as an example, amplitude of the load was between 2.0×10^7 - 7.0×10^7 N for two nodes cases, 1.0×10^7 - 5.0×10^7 N for three nodes cases, and 1.0×10^7- 4.0×10^7 N for four nodes cases.

Training data

When applying $10^6 P_a$ above magnitude force to the cantilever beam, it will have permanent plastic deformation. One of particular plastic displacement examples is as shown in Fig.5.3, while the associated plastic residual strain distribution is shown in Fig.5.4.

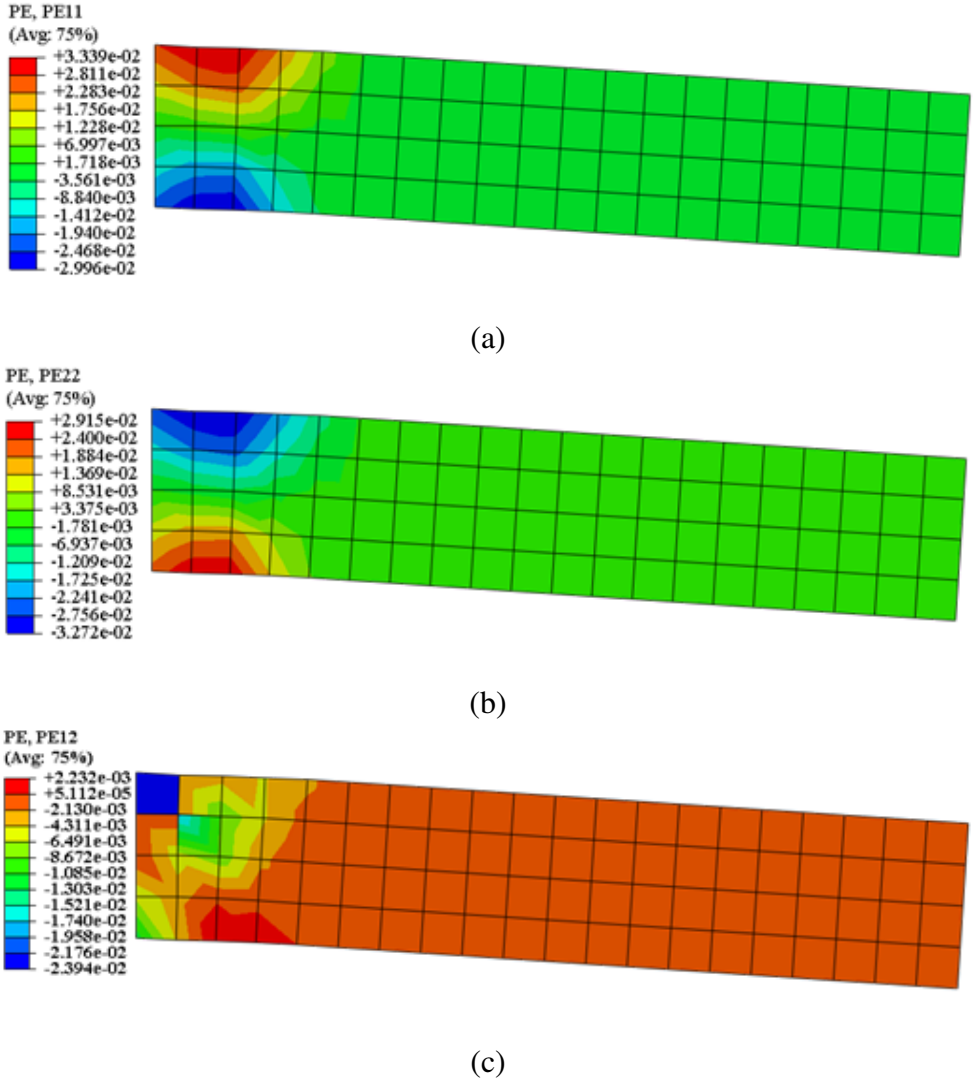


Figure 5.4: Permanent plastic strain distribution: (a) Plastic strain component ϵ^p_{11}, (b) Plastic strain component ϵ^p_{22}, and (c) Plastic strain component ϵ^p_{12}.

After obtaining the residual plastic displacements from all FEM nodes of the beam, we combine them together as the input of training data of DNN. The process is as shown in Fig.5.5. The right table of Fig.5.5 is the input of one set of raw training data of DNN. The output of this set of training data is the value and location of the outside force according to this plastic displacement distribution, for static modes, and the location, magnitude and duration according to this plastic displacement distribution, for dynamic models.

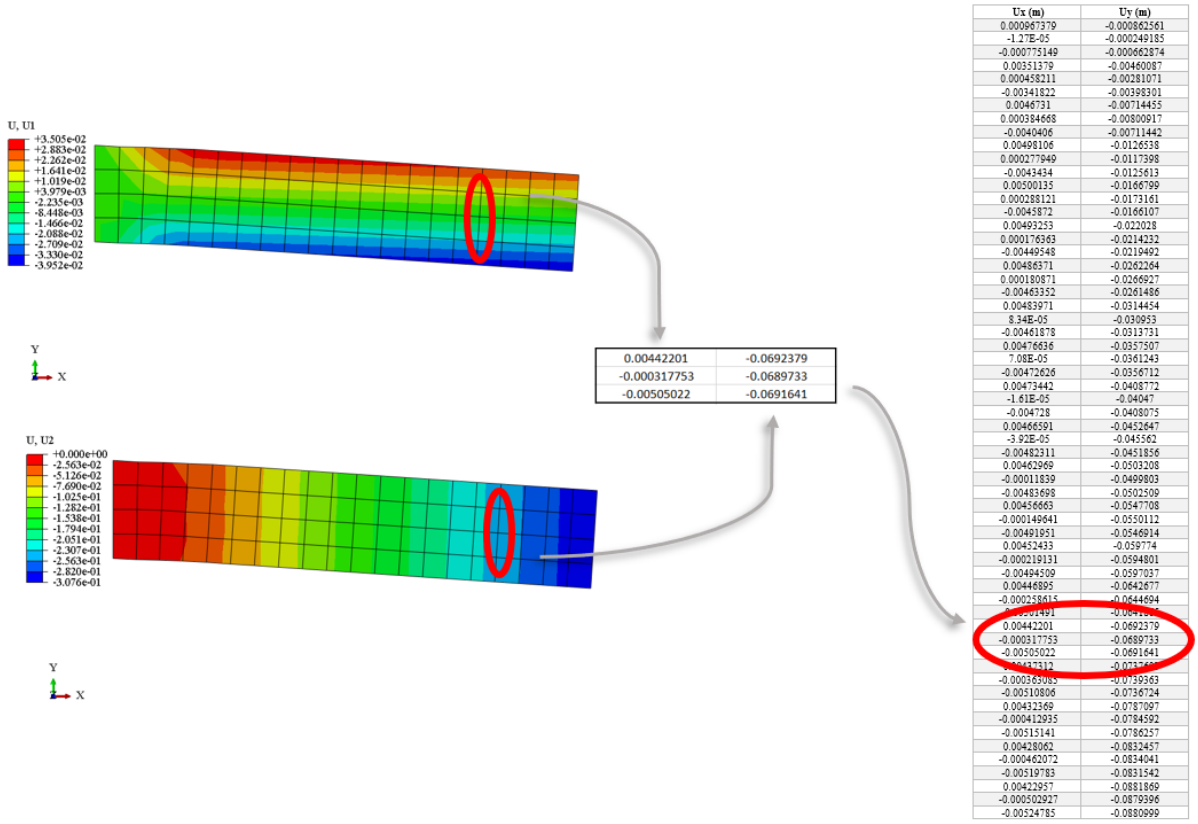


Figure 5.5: Collect one set of training data input

Results and Discussions

Prediction of the static loads on training nodes

In machine learning, the more the data, the more the prediction. However, there is still no rule about how much data is enough. It depends on how complex of the problem and how complex of the learning algorithm are. So the rules we used to generate data is as follows,

1. Generating the preliminary database with a small set of samples.

2. Training the model to see how the performance of the model is and if the predicted accuracy meets the requirement.

3. If the results are not accurate enough, then generating more data to see if the performance increases.

4. If the performance increases, then repeat the 1 to 3 steps until the result meets the required

Table 5.4: The correct and predicted loads of the testing cases $(10^7 N)$

Testing cases	**1**		**2**		**3**		**4**	
Combination	④		②④		①②④		①②③④	
	correct	predicted	correct	predicted	correct	predicted	correct	predicted
①x	0	-0.0353	0	-0.1221	2.1543	2.0785	1.2917	1.3145
①y	0	0.0214	0	-0.0524	-1.9067	-1.969	-1.5630	-1.5402
②x	0	-0.1143	4.5138	4.514	2.7296	2.8529	1.7999	1.8281
②y	0	-0.0335	-4.5320	-4.5488	-3.1083	-3.0752	-1.9380	-1.9136
③x	0	0.0294	0	-0.0004	0	0.0598	2.1204	2.1566
③y	0	-0.0469	0	0.0317	0	-0.0062	-2.7124	-2.8005
④x	9.4431	9.2816	6.1548	6.0149	3.4294	3.3453	2.8181	2.793
④y	-9.8902	-9.6926	-5.8801	-5.8765	-3.8756	-3.8447	-2.6192	-2.5966

Table 5.5: Predicted errors of the testing cases (%)

Loading Cases	**1**	**2**	**3**	**4**
Combination	④	②④	①②④	①②③④
①x	null	null	3.5182	1.7654
①y	null	null	3.2667	1.4555
②x	null	0.0041	4.5163	1.5689
②y	null	0.3699	1.0641	1.2582
③x	null	null	null	1.7097
③y	null	null	null	3.246
④x	1.7104	2.2732	2.4529	0.8908
④y	1.9981	0.0602	0.797	0.8644

accuracy.

5. If the performance stays still or increases slowly, then modifying the learning model or learning parameters.

For our static problem, we generated 290 sets of data (about 20 sets for each load case) from which we get a good training effect. The training effect is usually evaluated by the training loss which is a value representing the fitting of the model to the training data. In this study, the mean square error between the outputs and the correct results was chosen as the training loss. Minimizing the loss is the goal of the training process. There is an updated loss in each epoch which can show the fitting of the model. The record of the loss throughout the process of our case is shown as follows:

In Fig.5.6, the loss gradually decreases and reaches the minimum value of 0.2611 after 18k steps. The minimum value and the smooth descend curve of the loss indicates that the model was

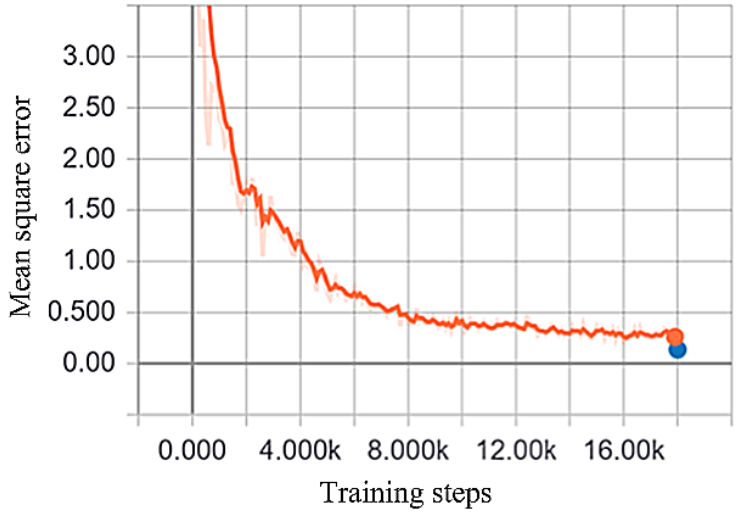


Figure 5.6: Training loss for prediction of the static loads on training nodes

trained steadily and has fitted the training data very well.

To test the performance of the model, four sets of testing data were generated. Each testing data represents one type of loading combination, as shown in Table 5.4. For this problem, we designed our output layers with eight neurons which represent the two direction loads in four valid loading points, as mentioned above. Therefore, inputting the testing data to our DNN model, we got the output as shown in Table 5.4. The errors of the prediction are listed in Table 5.5.

In Table 5.4, due to the eight-neuron output layer, all the eight values of the output were non-zero, but the values on the real-loaded nodes were much larger than the values on the other nodes. For example, in case 1, the real load is located in node ④. In the predicted result, the values of ④x and ④y is obviously much larger than the values on the other nodes. And the values of ④x and ④y is very close to the real values, with the errors of 1.71% and 1.998%. The output of the other three test cases have the same pattern. Predicted errors are all smaller than 5%, as shown in Table 5.5. Therefore, the trained DNN model can correctly predict the loading locations and the magnitude of the static loads which act on the training nodes.

Prediction of the static loads between training nodes

Section 5.1 had demonstrated the capability of the DNN model to predict the statics loads acting on the training nodes. In this section, following the data collection rules in Section 5.1, we trained the DNN model with 133 sets of data which were caused by static loads acting on the four valid training nodes individually. Then, we tested the model with 8 sets of deformation caused by 8 different loads acting in the interval between the training nodes, to see if the DNN can make extended prediction. The testing data is shown in Table 5.6. We designed the output layer with

Table 5.6: Correct values of the testing data

Testing Cases	1	2	3	4	5	6	7	8
Interval	③∼④	③∼④	②∼③	②∼③	①∼②	①∼②	①∼②	①∼②
Location (m)	3.0	3.25	3.75	4.0	4.5	4.5	4.75	4.75
Load x $(10^7 N)$	12	11.47	10	9.02	6	6.45	6.34	7.34
Load y $(10^7 N)$	-12	-10.35	-10	-9.56	-6	-6.63	-6.76	-7.76

Table 5.7: Predicted results of the testing data

Testing Cases	1	2	3	4	5	6	7	8
Interval	③∼④	③∼④	②∼③	②∼③	①∼②	①∼②	①∼②	①∼②
Location (m)	2.922	3.037	3.82	4.131	4.723	4.543	4.835	4.618
Load x $(10^7 N)$	12.214	11.163	9.327	8.356	5.675	6.389	5.886	7.167
Load y $(10^7 N)$	-12.065	-10.808	-9.9	-9.383	-6.108	-6.707	-6.599	-7.877

Table 5.8: Predicted errors of the testing data (%)

Testing Cases	1	2	3	4	5	6	7	8
Interval	0	0	0	0	0	0	0	0
Location	2.604	6.554	1.88	3.281	4.949	0.96	1.785	2.769
Load x	1.784	2.68	6.729	7.366	5.418	0.814	7.159	2.361
Load y	0.545	4.424	1.002	1.85	1.792	1.164	2.377	1.505

three neurons which represent the location of the load and the magnitudes of the loads in x and y directions because the loads are all acting individually in this section. The predicted results are shown in Table 5.6 and Table 5.7.

According to the results, all the predicted locations were in the correct intervals. The correct interval is an important information which can indicate the location range of the load. Then one can further locate the load by subdividing or reducing the interval. For the prediction of the specific location and the magnitude in x and y directions, the maximum errors are 6.554%, 7.366% and 4.424% respectively. The average error of the output is 3.098%, which is less than 5%. The errors of the prediction are small and can meet a lots of engineering requirements. Therefore, the results show that the DNN model is able to predict the static loads in the interval between the training nodes.

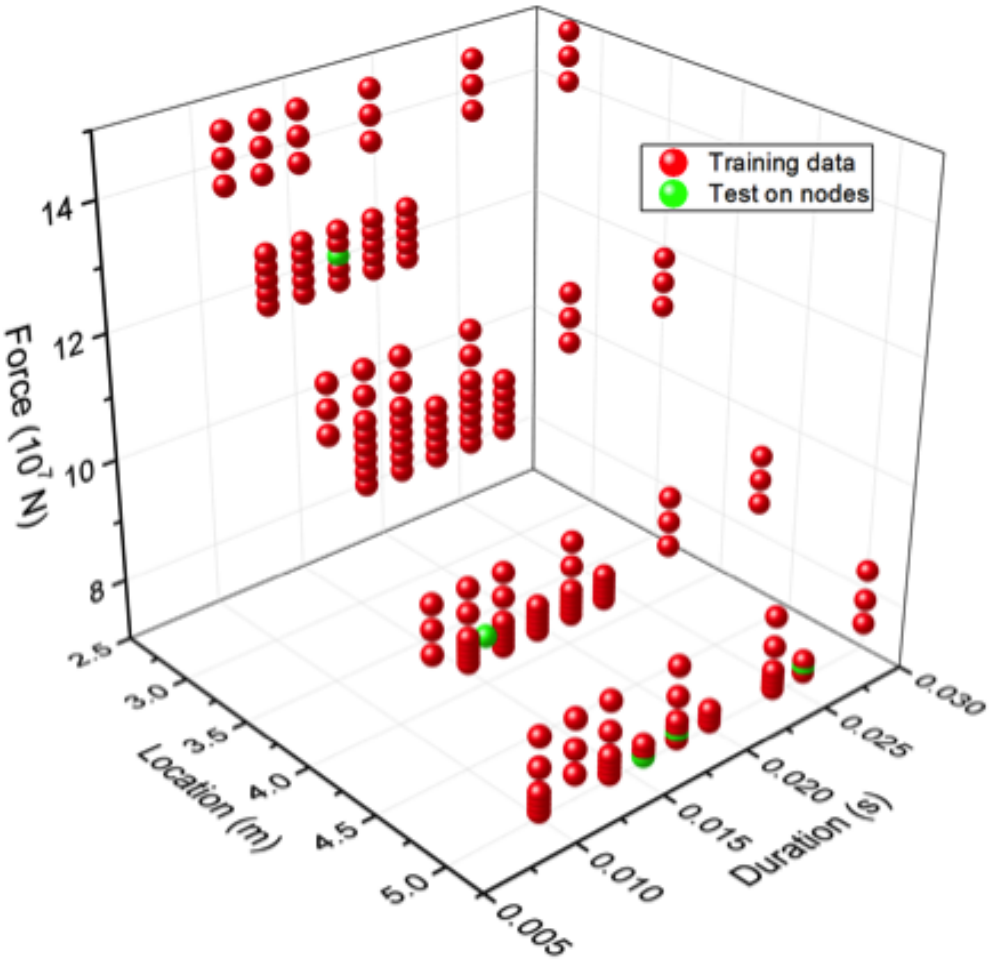


Figure 5.7: Training and testing data

Table 5.9: Predicted errors of the nodal impact loads(%)

Loading Cases	1	2	3	4
Location	3.321	0.94	3.175	0.474
Magnitude	0.289	0.293	1.641	2.338
Duration	12.321	5.032	0.682	3.535

Prediction of the impact loads on training nodes

Compared to the static problems, dynamic problems are more common in real situations but also more complicated. By following the data collection rules in Section 5.1, for our dynamic problem here, about 40 sets of data for each load case would produce a good training effect. We finally trained the DNN model with 175 sets of deformation caused by different single vertical

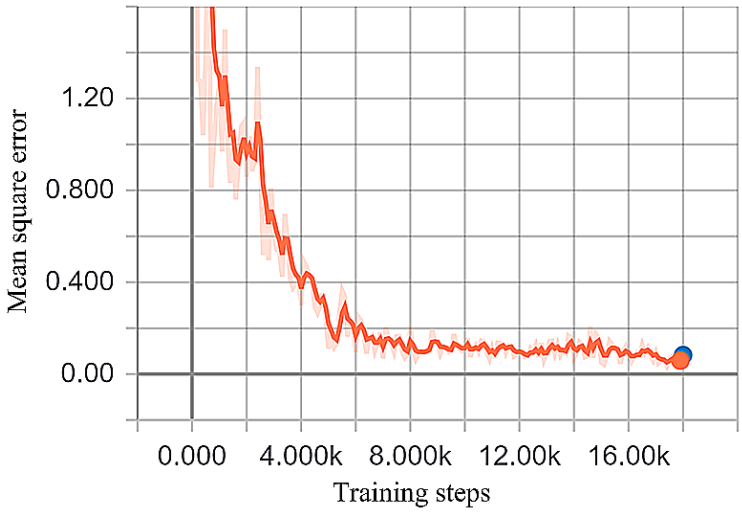


Figure 5.8: Training loss for prediction of the impact loads on training nodes

impact load acting on the training points. Then five sets of test data were generated to test the performance of the trained model. All the training and testing data are shown in Fig.5.7. The output layer has three neurons which represent the location, magnitude and the duration of the impact loads.

The training loss throughout the training process is shown in Fig.5.8. The minimum loss (MSE) reached 0.056 after 18k iteration steps which shows a good fitting of the model to the training data. The prediction of these impact loads are shown in Fig.5.9. The predicted loads and the correct loads are very close to each other in the shown three-dimensional space. The maximum errors of the predicted location and the magnitude are 3.321% and 2.338%, respectively, as shown in Table 5.9. The maximum error of the duration is 12.321%. During the study, we found that the duration is more difficult to be predicted than the other two parameters, possibly because of the increase of complexity of the behavior due to dynamic loading. The overall errors of these five testing cases are less than 5.4%. It shows that the DNN also works very well in the prediction of the impact loads which are located on the training nodes, especially for the location and the magnitude of the impact.

Prediction of the impact loads between training nodes

For the prediction of the impact loads acting within different intervals, we generated three sets of testing deformation which were caused by three different impact loads acting in each interval individually. Then, we used the DNN model in the previous section which had been trained with the 175 sets of deformation by single impact loads acting on training nodes. The values of the training data and the test loads are shown in Fig. 5.10. The output of the DNN model is shown in Fig.5.11, and the error of the output is listed in Table 5.10. The maximum error of the location,

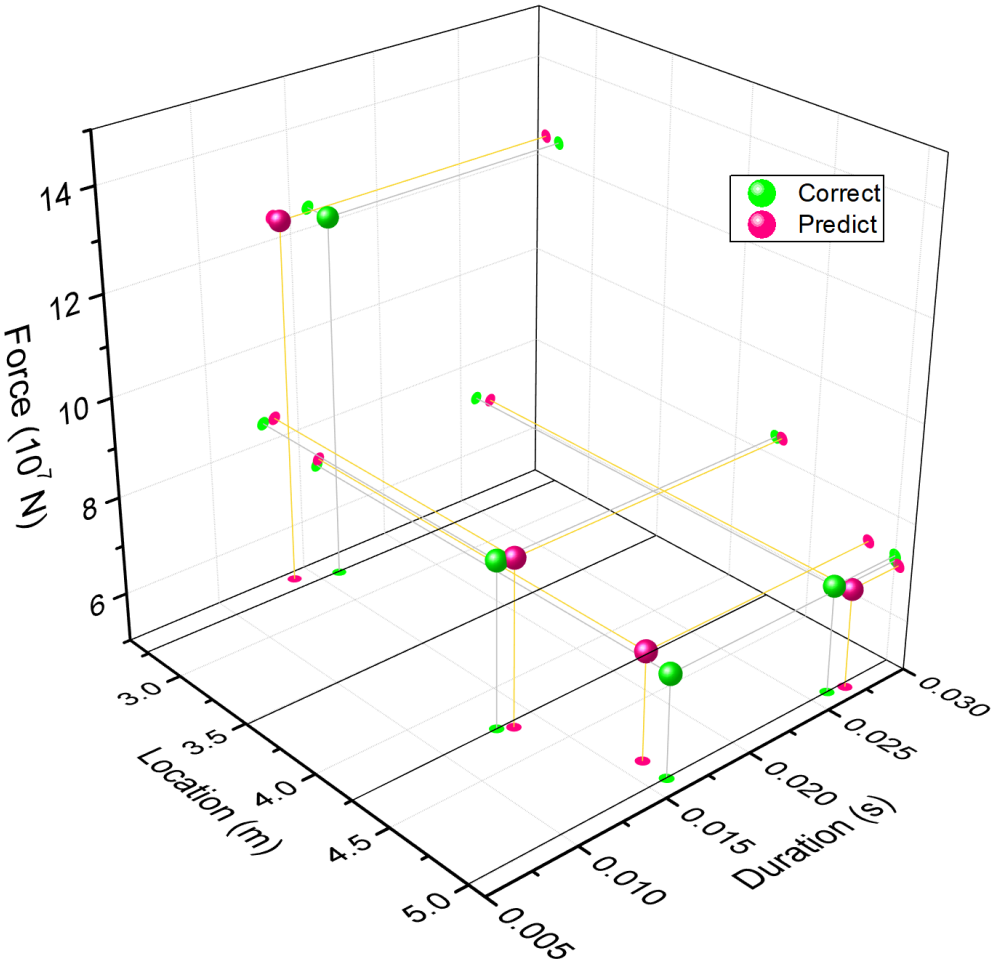


Figure 5.9: Predicted result of the loads on the nodes

magnitude and duration are 9.718%, 6.869% and 7.213%, respectively. Compared to previous section, the overall errors are increased but still less than 10%. And the predicted locations are all in the correct intervals. It is concluded that the DNN can predict the loads in the interval between the training nodes, too.

The results show that the DNN model is capable of making an interpolation between the training data by itself. But the interpolation accuracy still depends on the density of the training data. The higher the density of the training data, the higher the predicted accuracy will be. So while training the DNN model, we can choose some typical loading cases as training data depending on the accuracy demand, which would save a lot of time in training phrase. To further reduce the predicted error, another approach is developed as explained in the next section.

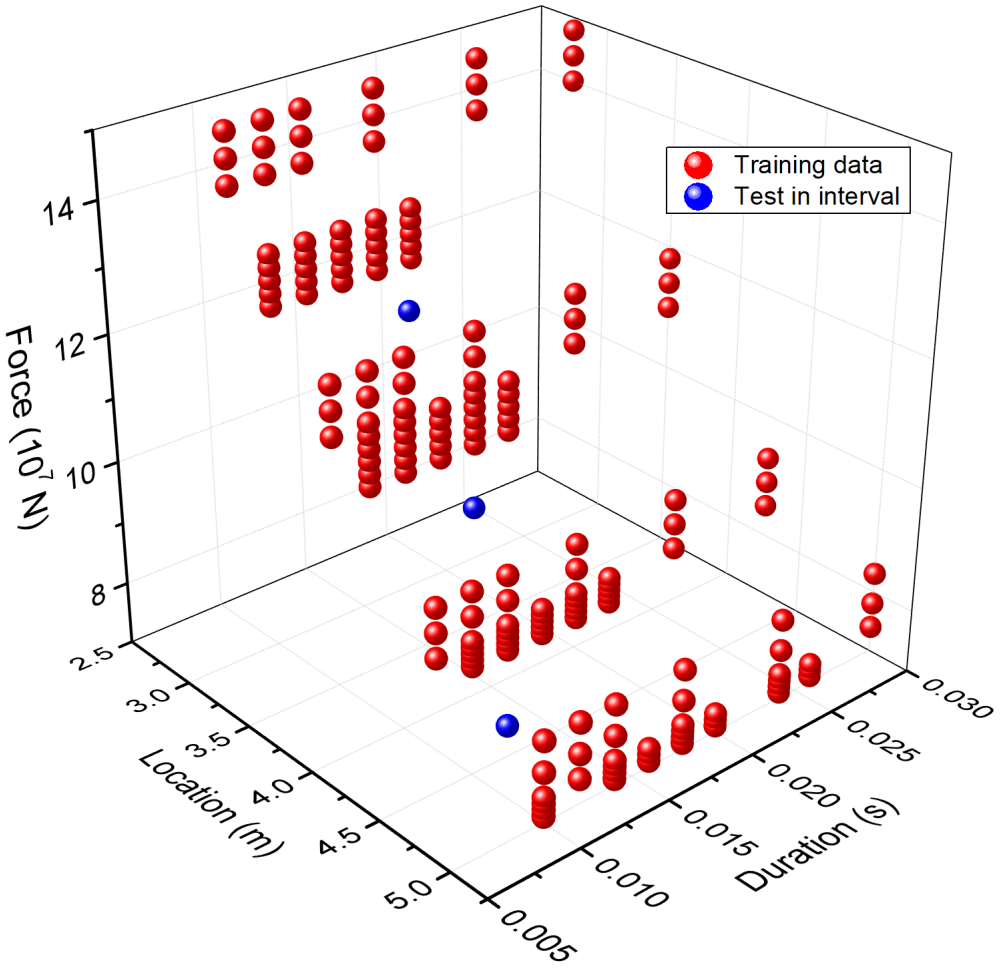


Figure 5.10: Training and testing data

Improving accuracy of prediction on the load location

According to the above results, it is almost for sure that the DNN model can predict the interval of the load location correctly. Therefore, after the first prediction, we can concentrate our attention to the predicted interval. To further locate the true value of the load, we can subdivide the interval into several smaller intervals, and then add more training data within this area. Here, we took the Testing Case 3 in Table 5.6 and Table 5.7 as an example. The DNN model had predicted that the load was located in interval ①∼②. So we can subdivide the interval ①∼② into three smaller intervals. The location coordinates of the smaller intervals are 4.25∼4.375m, 4.375∼4.625m and 4.625∼5m, as shown in Fig.5.12. Eighteen sets of new training data were generated on each of the new numbered points ⑤ and ⑥, respectively. Then, we retrained the DNN model with the data on Node ①,②,⑤ and ⑥. Different from the previous model which was trained with all data, the retrained model would only predict the loads in interval ①∼② and achieve a much higher

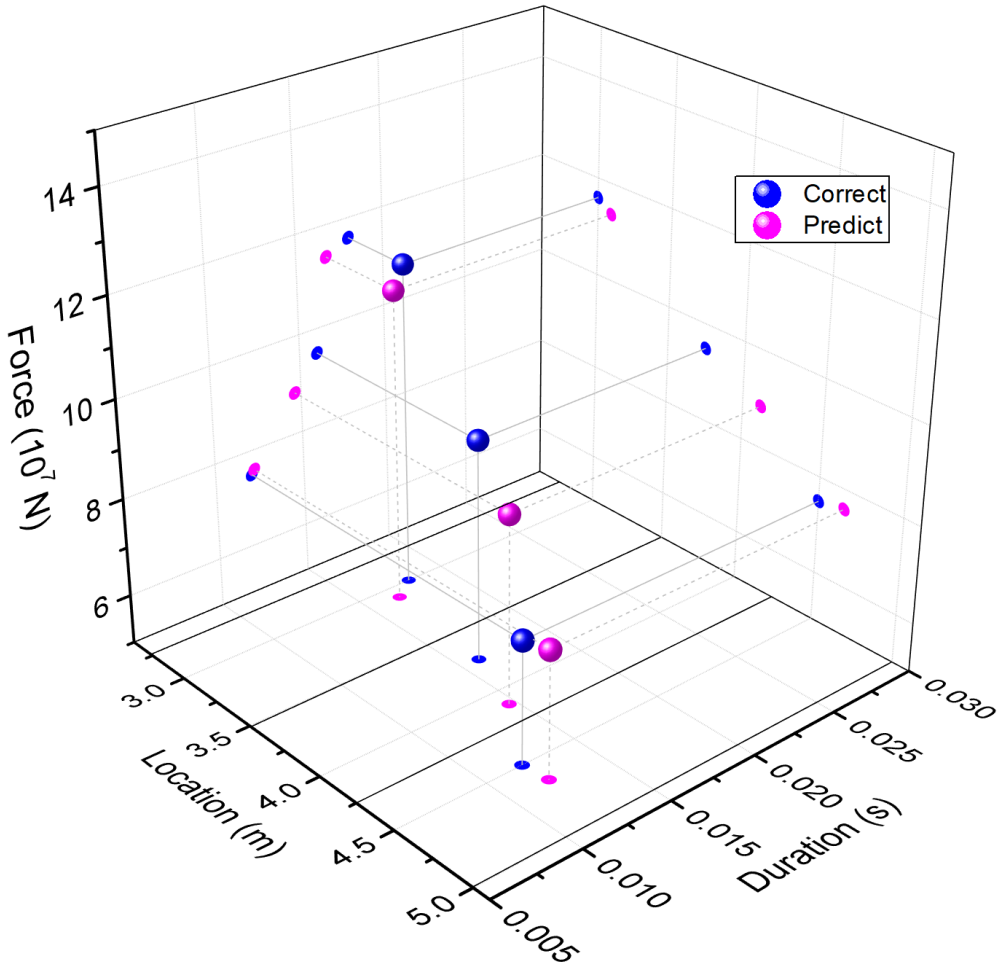


Figure 5.11: Predicted result of the loads in the intervals

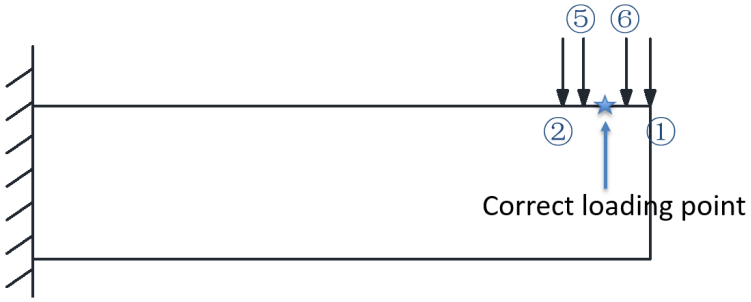


Figure 5.12: Refined interval

accuracy in the prediction. Two testing data were generated, as shown in Table 5.11. The predicted results and the errors before and after the retraining are shown in Table 5.12 and Table 5.13. The

Table 5.10: Predicted errors of the loads in intervals(%)

Predicting Cases	1	2	3
Interval	③∼④	②∼③	①∼②
Location	3.058	9.718	3.418
Magnitude	2.047	6.859	1.195
Duration	7.213	8.516	1.233

Table 5.11: Correct values of the predicted impact loads

Predicting Cases	1	2
Interval	⑤∼⑥	⑤∼⑥
Location (m)	4.5	4.5
Magnitude($10^7 N$)	8.2	8.4
Duration (s)	0.018	0.014

Table 5.12: Predicted value of the impact loads

Predicting Cases	1	2	1	2
	Before refine		After refine	
Interval	⑥∼①	⑥∼①	⑤∼⑥	⑤∼⑥
Location(m)	4.906	4.66	4.464	4.444
Magnitude($10^7 N$)	7.332	7.739	8.444	8.435
Duration(s)	0.0191	0.0146	0.0171	0.0145

predicted errors of all three parameters are reduced greatly by refining the interval, from 9.02%, 10.583% and 5.945% to 0.798%, 2.978% and 4.864%, respectively. Also, the prediction of the smaller intervals reached 100% accuracy. Therefore, by refining the interval and retraining the DNN model step by step, it is possible to finally reach the required accuracy.

From the tendency, it is also predicted that finer mesh will lead to correct finer intervals and high accuracy results. Thus, if we could have large amounts of data with forces applied on a large variety of different locations, which is equivalent to dividing the cantilever beam to large numbers of sections, the predicted results will still be in the correct interval. The accuracy will achieve almost 100% if we have unlimited amount of data.

Table 5.13: Predicted errors of the nodal impact loads(%)

Predicting Cases	1 Before refine	2	1 After refine	2
Interval	⑥∼①	⑥∼①	⑤∼⑥	⑤∼⑥
Location	9.02	3.552	0.798	1.238
Magnitude	10.583	7.866	2.978	0.421
Duration	5.945	4.236	4.864	3.725

5.2 Elasto-plastic Shell Structures

Data Collection

Finite element model

The failure process that we are modeling in this study is the contact collision between a hemispherical shell with a 3D rigid cylindrical body. The loading parameters that are needed to predicted are: (1) Contact location of the two bodies, (2) The contact velocity, and (3) The contact duration. The observed or measured input data are permanent plastic deformation of the hemispherical shell at each finite element nodal points. The radius and thickness of the hemisphere are 1 m and 5 mm, respectively. Radius and length of the rigid cylindrical body are 0.08 m and 0.18 m, respectively. The initial distance between the hemispherical shell and the rigid cylindrical body is set as 0.01 m. The finite element model of the hemispherical shell and the rigid cylindrical body were developed by using the commercial software ABAQUS, as shown in Fig. 5.13.

The hemispherical shell was modeled by using S4R element, which is a four-node membrane element [113]. The cylindrical body was modeled by using the eight-node linear solid element. For the hemispherical shell, the mesh size was 3.5 mm, which results in a total number of nodes of 6441 and a total number of elements of 6348. For the rigid cylindrical body, the total number of nodes and elements are 835 and 672, respectively.

In the modeling and simulation, we first identify and mark a reference point positioned at the center of the upper surface of the cylindrical body, as shown in Fig. 5.13. Then, the rigid cylindrical body with some initial velocity will collide with the hemispherical shell.

Both static and dynamic responses of the hemispherical shell are numerically calculated. In order to solve the convergence problem caused by the large deformation of the hemispherical shell, and the contact between hemispherical shell and the rigid body, Abaqus/Explicit is used to conduct all the analysis. The standard contact algorithm is used to simulate the contact between the hemispherical shell and the rigid cylindrical body.

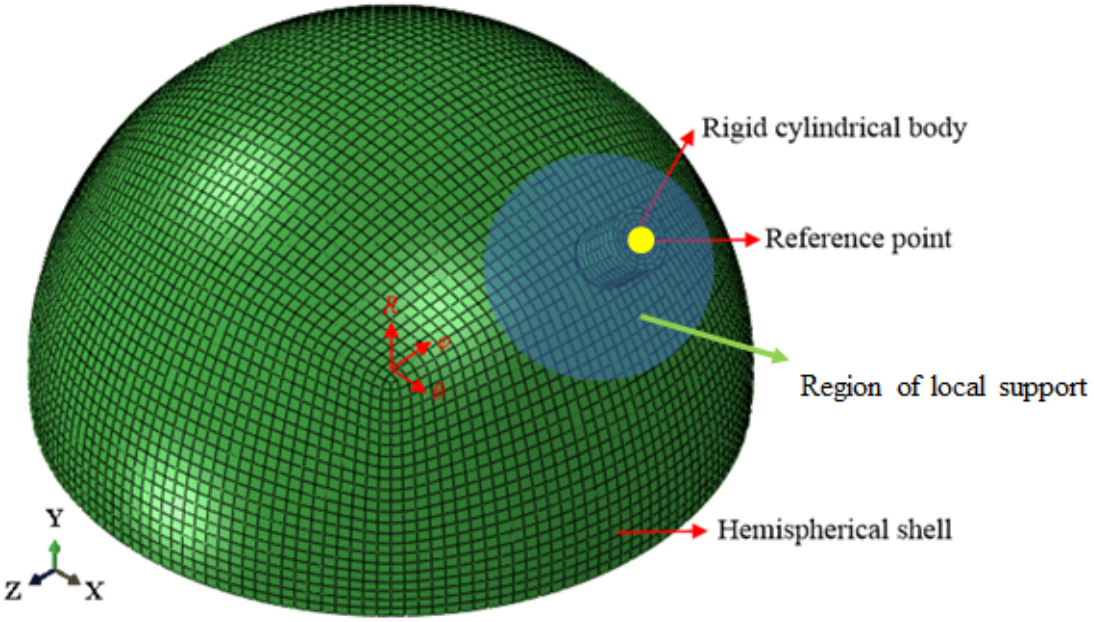


Figure 5.13: Finite element contact collision model of a hemispherical shell and a rigid cylindrical body

Material properties

The material of the hemispherical shell is chosen as AISI 4340 steel (33 HRc), and both plasticity and damage parameters of AISI 4340 are modeled by using the Johnson-Cook model [47, 63]. The flow stress of the Johnson-Cook model is expressed as follows,

$$\sigma = [A + B(\epsilon^p)^n][1 + Cln(\frac{\dot{\epsilon}^p}{\dot{\epsilon}_0})][1 - (\frac{T - T_r}{T_m - T_r})^m] \tag{5.4}$$

where A, B, C, m, n are material constants, ϵ^p is the plastic strain, $\dot{\epsilon}^p$ is the strain rate, $\dot{\epsilon}^0$ is the reference strain rate, T is the absolute temperature, T_m is the melting temperature, and T_r is the reference temperature.
The critical failure strain ϵ^f is expressed as following [47],

$$\epsilon_f = [D_1 + D_2 exp(D_3 \frac{\epsilon_p}{\epsilon_e})][1 + D_4 ln(\frac{\dot{\epsilon}^p}{\dot{\epsilon}_0})](1 + D_5 \frac{T - T_r}{T_m - T_r}) \tag{5.5}$$

where D_i are material constants, ϵ_p is the pressure stress, and $\epsilon_e$is the von Mises stress.
All material parameters of the AISI 4340 steel (33 HRc) are listed in Table 5.14, where ρ is the material density, E is the Youngs modulus, υ is the Poisson ratio.

Initial and boundary conditions of the collision

For the hemispherical shell, all nodes at y=0 m were rigidly fixed. For the rigid cylindrical body, the initial condition was applied to the reference point in the spherical coordinate system

Table 5.14: Material parameters of the elasto-plastic shell structures

Elastic properties		Plastic properties		Damage properties	
ρ	7830 kg/m3	A	792 MPa	D1	0.05
E	208 GPa	B	510 MPa	D2	3.44
υ	0.3	C	0.014	D3	-2.12
		m	1.03	D4	0.002
		n	0.26	D5	0.61

(R, θ, ϕ), as shown in Fig. 5.13. It is worth noting that the spherical coordinate system was determined based on the initial position of the reference point, and the θ axis is always parallel to the line connecting the center of the sphere and the projection point of the reference point in the XZ plane. Because the rigid cylindrical body was assumed to collide with the hemispherical shell at the points with different longitudes and latitudes, the spherical coordinate system is not fixed. Except the radial distance and the polar angle, all degrees of freedom of the reference point were rigidly fixed. Both radial velocity V_1 and polar velocity V_2 were applied to the reference point in the spherical coordinate system (R, θ, ϕ), and the two velocities had the same amplitude, which is featured with a loading and unloading and can cause plastic deformation of the hemispherical shell, as shown in Fig. 5.14. As mentioned above, in order to solve the convergence problem caused by the large deformation and the contact between hemispherical shell and the rigid body, Abaqus/Explicit quasi-static simulation is used to conduct the static analysis, so the duration t in static analysis was set as 0.5 s and the step time T_{max} equals 1. On the other hand, during dynamic collision, the duration of the contact t was set as 0.01 s, 0.015 s, 0.02 s, respectively, and T_{max} equals 0.1 for the dynamic analysis. The machine learning algorithm is used to predict the first contact point of the hemispherical shell, which is closely related to the initial position and velocities of the rigid cylindrical body, and the radial and polar velocities of the rigid cylindrical body. For the dynamic analysis, the duration of the velocities of the rigid body should also be predicted by the machine learning algorithm. For this purpose, by assuming that the rigid cylindrical body positioned at different locations and moved at different velocities towards the hemispherical shell. Then, the responses of the hemispherical shell were analyzed, and the final plastic deformations of the shell were collected. Loads and durations for the numerical analysis are listed in Table 5.15.

The initial positions of the rigid cylindrical body were assumed at different latitudes and longitudes of the hemispherical shell. Five different latitudes were considered: $25°$, $45°$, $60°$, $75°$, $90°$, and five longitudes were considered: $0°$, $22.5°$, $45°$, $67.5°$, $90°$. Intersections of the latitudes and longitudes are the initial position of the rigid body, as shown by purple circles in Fig. 5.15. The initial distance between the hemispherical shell and the rigid cylindrical body was constant for all case studies. Finally, there are 525 sets of data collected for the static analysis, and 1575 sets of data for the dynamic analysis.

Table 5.15: Loads and duration for the static and dynamic analysis

Load cases	Static analysis			Dynamic analysis		
	V_1 (m/s)	V_2 (m/s)	t (s)	V_1 (m/s)	V_2 (m/s)	t (s)
1	0.2 - 0.4	-0.20	0.5	10 - 12	-10.0	0.01, 0.015, 0,02
2	0.2 - 0.4	-0.25	0.5	10 - 12	-10.5	0.01, 0.015, 0,02
3	0.2 - 0.4	-0.30	0.5	10 - 12	-11.0	0.01, 0.015, 0,02
4	0.2 - 0.4	-0.35	0.5	10 - 12	-11.5	0.01, 0.015, 0,02
5	0.2 - 0.4	-0.40	0.5	10 - 12	-12.0	0.01, 0.015, 0,02

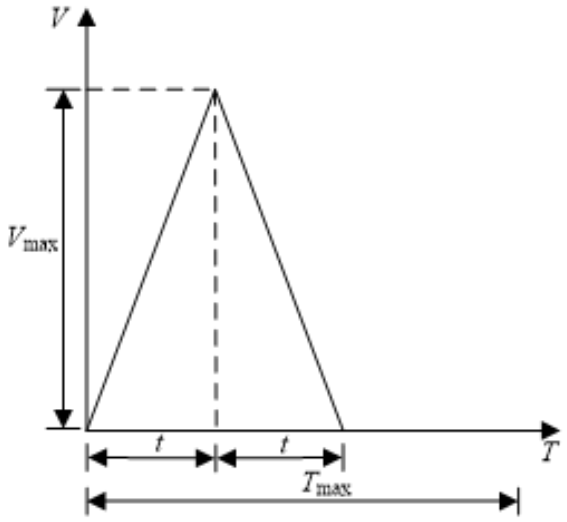


Figure 5.14: Velocity-time curve for the static and dynamic analysis

Numerical Case Studies

In order to verify and validate the accuracy and effectiveness and performance of the proposed data-driven approach, three different case studies were performed, which are discussed in this section. The three cases studies contain: quasi-static and dynamic loads acting on (I) training points, on (II) the points that are in the intervals between training point, and on the points that are outside of the training area.

Damage loads on training points

To test the capability of the proposed approach in predicting the damage loads on training points, we generated five plastic deformations caused by two static loads and three dynamic collision loads. All testing cases are not within the training data set, but the collision contact points are

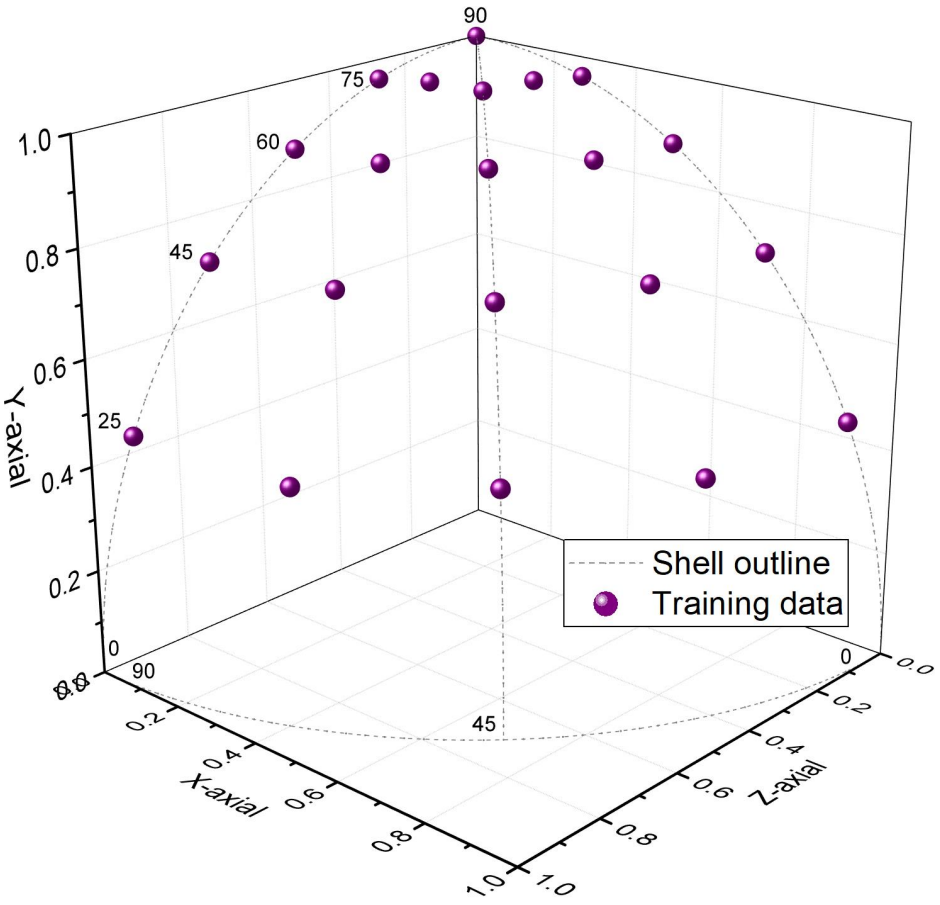


Figure 5.15: Different initial longitude and latitude position of the rigid cylindrical body

the training points. The summary of testing cases are shown in Table 5.16. Following the flowchart, the high-dimension testing deformation should first be input to PCA module to reduce dimensions. For the data goes to DNN1, we set the PCA parameter as 0.95 for static loads and 0.99 for dynamic loads, which means 95% and 99% of the variance is retained in the low dimension data set. In this case, 95% and 99% of the variance amounts to 10 and 19 principle components. Therefore, each set of testing deformation which contains $18k$ dimension will be reduced to a matrix with 10 and 19 dimensions. The whole training data set has 525 sets of static deformation and 1575 sets of dynamics deformation. Thus, the finial sizes of the training data matrix are $525*10$ and $1575*19$.

For this case, we adopted a neural network with two hidden layers. The first hidden layer has 128 neurons and the second one has 8 neurons. The hyper-parameters of our model are listed in Table 5.17. The initial weights of the neural network is generated randomly by TensorFlow default setting, and it is changing in the learning process continuously. The location and duration were predicted first. The training history of the DNNs for location and duration is shown in Fig. 5.16, Fig. 5.17 and Fig. 5.18.

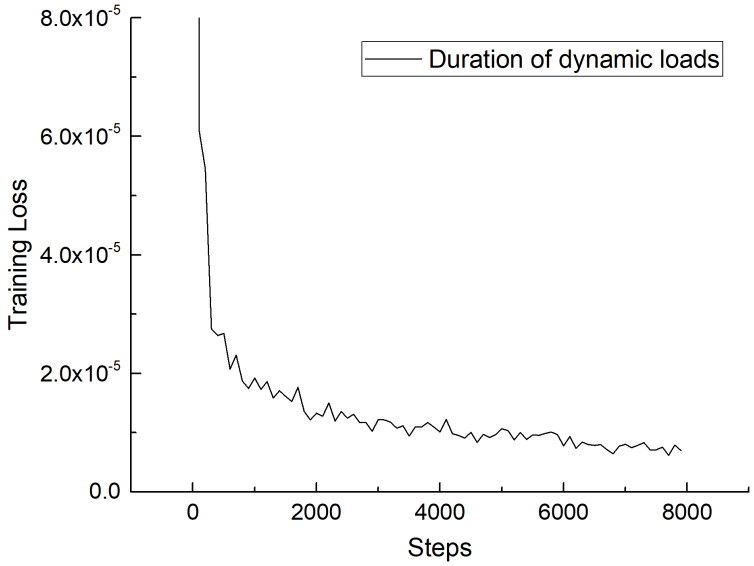


Figure 5.16: Training loss of the duration of dynamic cases

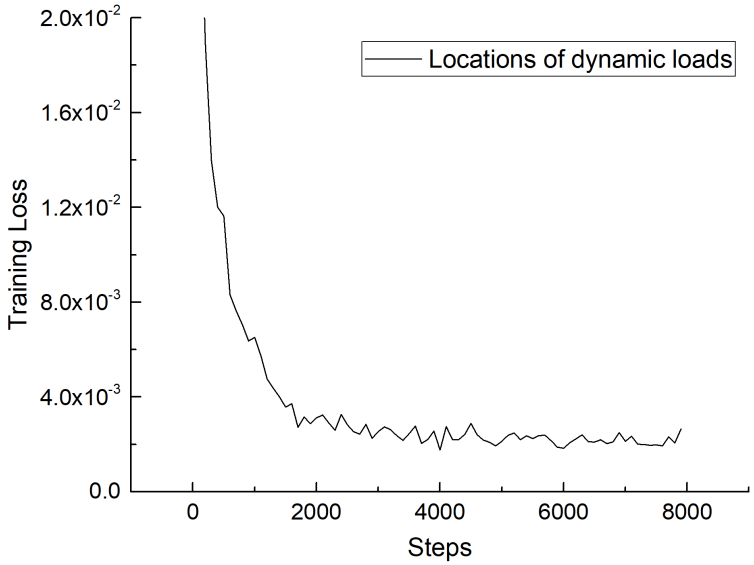


Figure 5.17: Training loss of the location of dynamic cases

In order to guarantee the stability of the prediction, the neural network has been trained and made prediction 10 times for each testing case. The predicted locations of the testing cases are shown in Fig. 5.19. The maximum errors of the X, Y, Z coordinates are all within 5%. The mean error of the static location and dynamic location are 1.0779% and 0.67875%, respectively. It shows that the neural network can stably and correctly distinguish the impact point out of 21

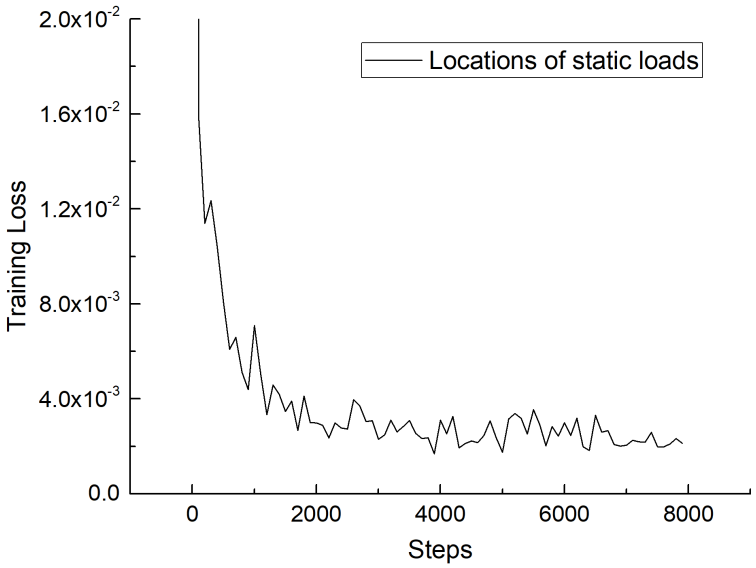


Figure 5.18: Training loss of the location of static cases

Table 5.16: Information of testing cases on training points

Testing cases	Static 1	Static 2	Dynamic 1	Dynamic 2	Dynamic 3
Longitude (°)	22.5	45	45	67.5	90
Latitude (°)	60	45	45	60	45
Location x	0.46	0.51	0.51	0.18	0
Location y	0.87	0.70	0.70	0.87	0.71
Location z	0.18	0.51	0.51	0.46	0.71
Normal speed (m/s)	0.25	0.35	10	11	12
Tangential speed (m/s)	0.25	0.35	12	11	10
Duration (s)	0.5	0.5	0.01	0.015	0.02

Table 5.17: Architecture and hyper-parameters of the neural network

Architecture / hyper-parameters	Values
Hidden layers	2
Neurons	128 / 8
Learning rate	0.0025
Learning rate decay rate	0.99
Dropout rate	0.025
Training steps	8000
Activation function	ReLU
Batch size	Full
Optimizer	Adam

Table 5.18: Predicted errors of loads acting on training points

	Static 1-2		Dynamic 1-3	
	Max Error	Mean Error	Max error	Mean Error
Location	2.04%	0.92%	1.27%	0.68%
Normal speed	2.00%	1.28%	2.37%	1.08%
Tangential speed	2.10%	1.34%	2.28%	1.47%
Duration	0%	0%	2.10%	0.86%

training points.

After predicting the location of the testing loads, the training data which is on the predicted location would be pull out to train DNN3 and DNN4 for predicting speeds. As mentioned in Section 3, we have generated 24 sets of training data for each Static Case 1 and 2. For the prediction of speeds, we set up two independent neural networks for normal speed and tangential speed. The architecture of the neural networks is the same as Table. 5.17. The PCA parameters here is 0.95 for static speed prediction and 0.999999 for dynamic speed prediction. The 10-times predicted results are shown as the blue circles and balls in Fig. 5.20 and Fig. 5.21. The errors are shown in Table. 5.18. The results show that the proposed approach can successfully predict the loads acting on training points.

Finally, with all the machine learning prediction values at hand, we calculated a mean value of each prediction, and make another FEM simulation with these values to recover the impact process of Case Dynamic 2. The comparison of the recovered result and original result is shown in Fig. 5.23. The errors of the recovered stress magnitudes of all elements are within 5%.

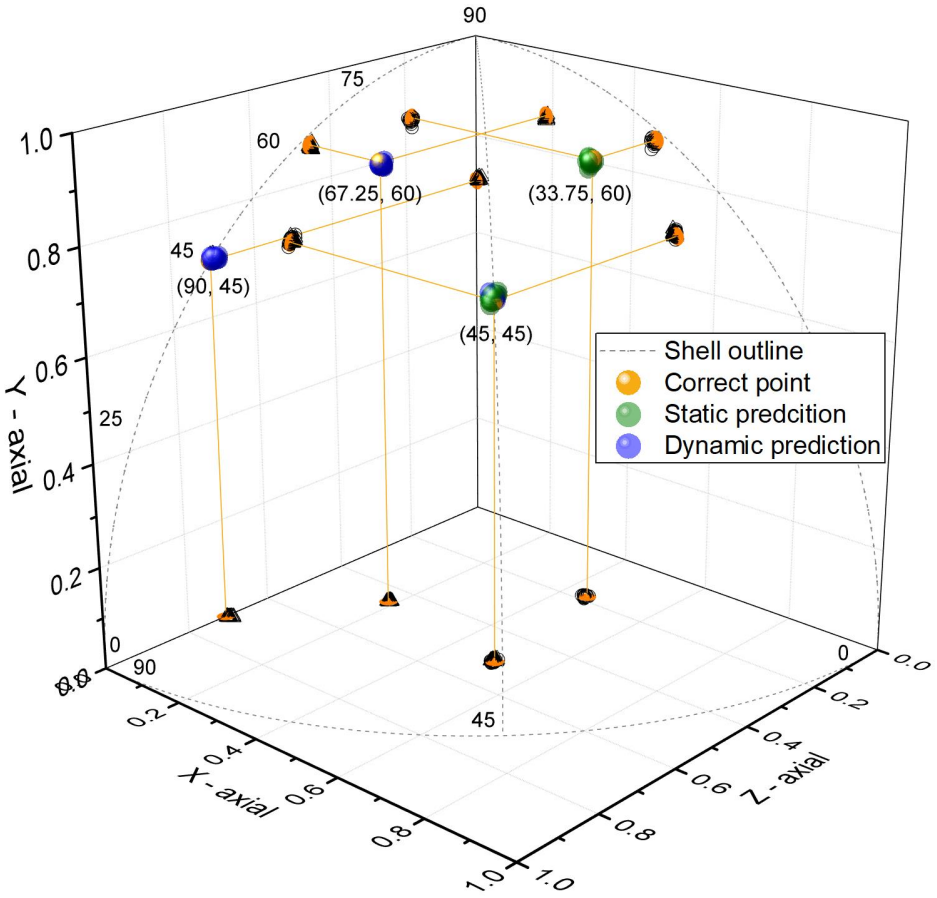


Figure 5.19: Predicted location of static and dynamic loads on training points

Collision points in the gap interval area among training nodes

In real situations of engineering applications, the impact or collision point must be randomly located on the hemisphere, and on the interval or gap area among training points. However, the permanent plastic deformation data caused by a collision that happened in the interval area are not contained in the training data set. Predicting a case not included in the training data is a challenge to every neural network. To test such ability of our model, we performed two testing cases in which collisions occur in the interval area between training point (22.5,45), (22.5,60), (45,45), (45,60). The details of the testing load conditions are shown in Table 5.19. In these cases, the neural network is trained with the same training data as mentioned above, which does not contain any data in the interval area. Following the flowchart in Section 2, DNN1 and DNN2 are set up, and are used for predicting the collision location and collision time duration. The PCA parameters for static and dynamic data are 0.95 and 0.99, respectively.

In order to guarantee the stability of the neural network, we trained the network for 10 times

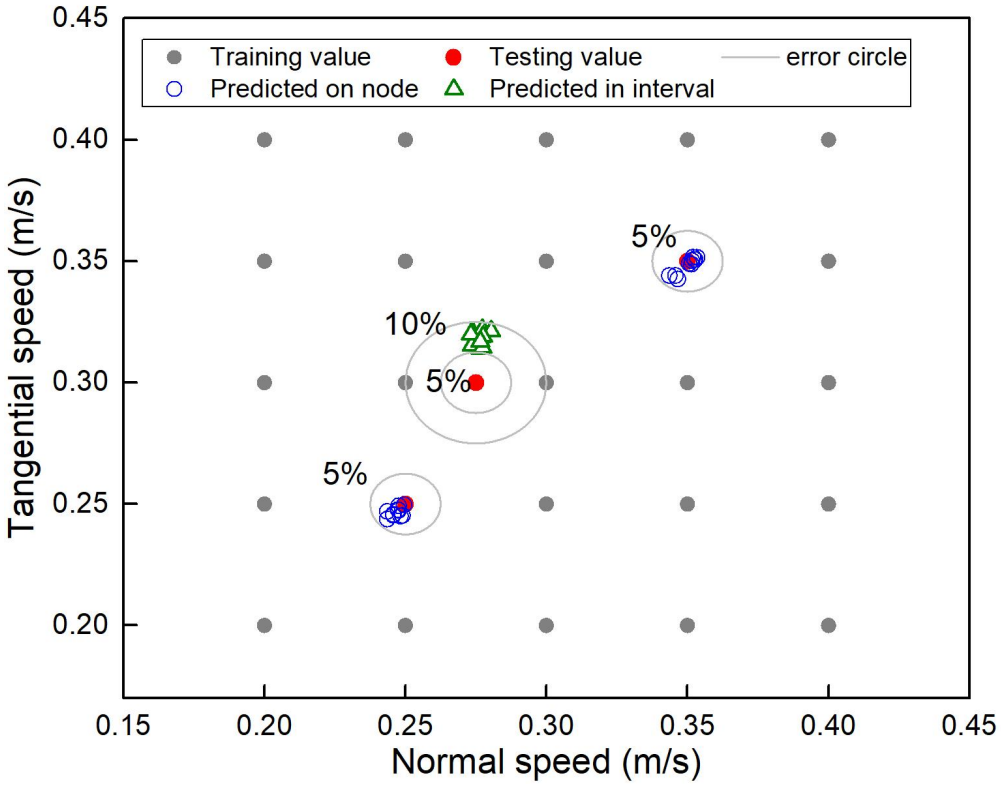


Figure 5.20: Predicted results of quasi-static speeds

Table 5.19: Information of testing cases in interval

Testing cases	Static 3	Dynamic 4
Longitude (°)	33.75	55
Latitude (°)	33.75	55
Location x	0.47	0.47
Location y	0.82	0.82
Location z	0.33	0.33
Normal speed (m/s)	0.275	10.5
Tangential speed (m/s)	0.3	10.5
Duration (s)	0.5	0.01

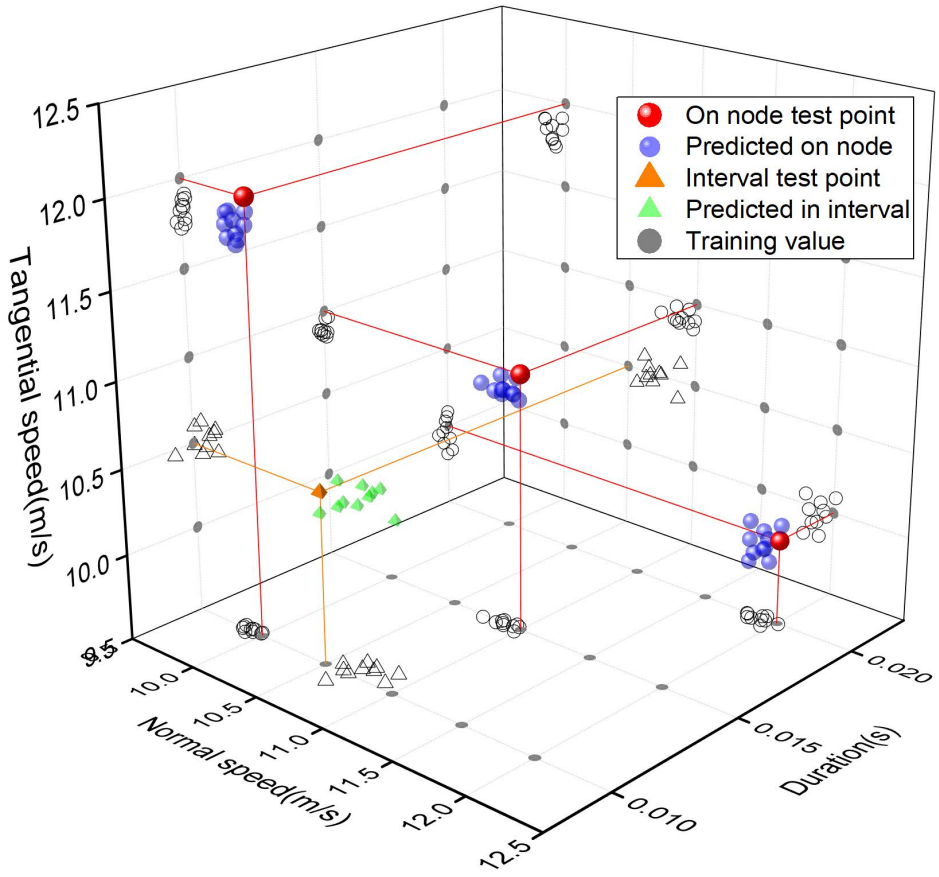


Figure 5.21: Predicted results of dynamic speeds and duration

Table 5.20: Predicted errors of loads acting in interval

	Static 3		Dynamic 4	
	Max Error	Mean Error	Max error	Mean Error
Location	16.82%	11.64%	15.02%	8.41%
Normal speed	1.97%	0.81%	3.71%	1.95%
Tangential speed	7.27%	6.08%	1.04%	0.57%
Duration	0%	0%	8.8%	4.86%

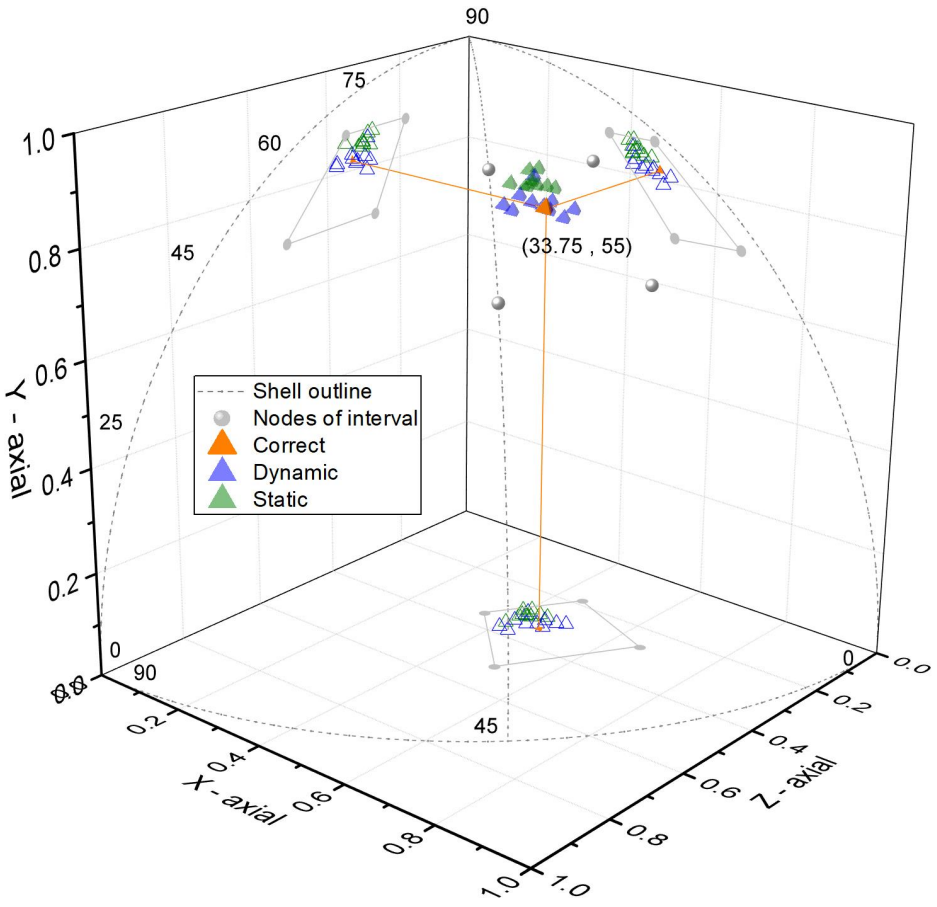
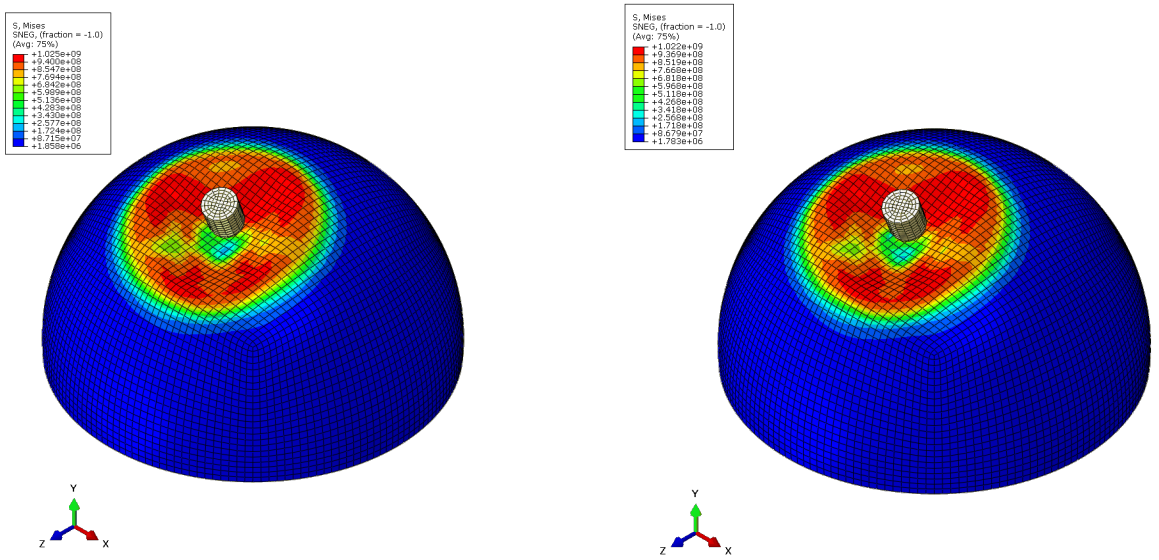


Figure 5.22: Predicted locations in interval

and made 10 predictions. The predicted results of the duration and location are shown in Fig.5.21 and Fig.5.22. The predicted errors are shown in Table. 5.20. The mean error of the 10-times prediction is 11.64% for static load and 8.41% for dynamic load. Even though the errors are larger than the errors in the previous section, all the predicted results are in the right interval. Thus, from the results here, we can figure out the interval that the loads acting on. Then, we can reduce the searching area and do further positioning within the interval.

Furthermore, for the prediction of collision speeds, we pull out the training data of the Point (22.5,45), (22.5,60), (45,45), (45,60) to train the next neural networks. The data for the neural network's prediction for collision speeds that we used here are the same as the ones in Table 5.17. The PCA parameters is 0.999999 for normal speed and 0.95 for tangential speed. The predicted speeds of the static load are shown as the green triangles in Fig. 5.20 and the dynamic one is shown as the green tetrahedrons in Fig. 5.21. The errors are listed in Table 5.20. All predicted speeds are within the 10% error circle and in the same interval area as the testing value.



(a) (b)

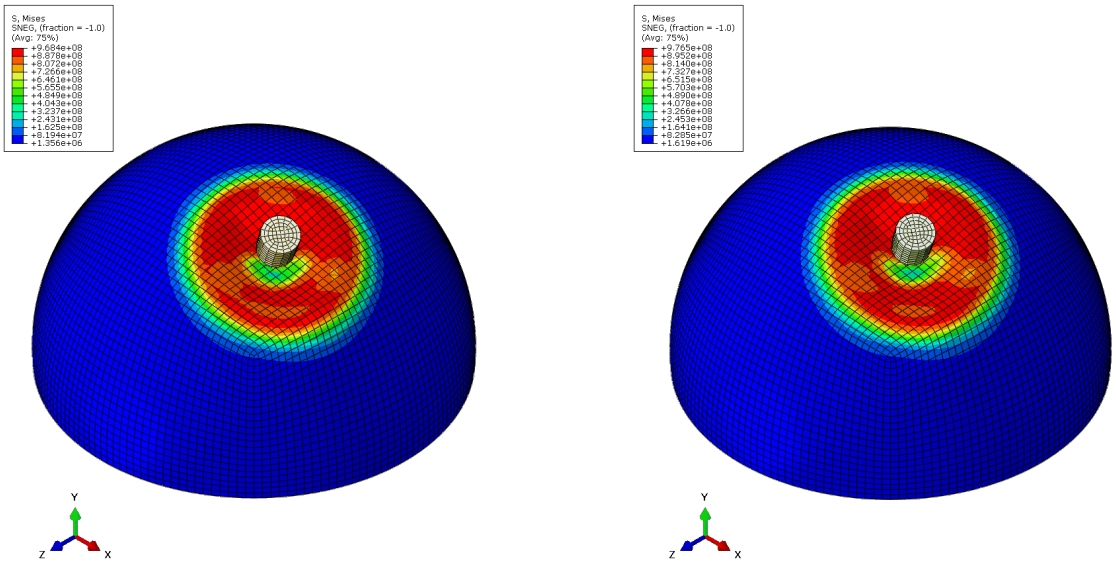
Figure 5.23: Stress distribution of Dynamics case 2: (a) stress distribution of test data; (b) recovered stress distribution generated by ML results.

Finally, we used the predicted damage load of Case Dynamic 4 to recover its impact collision process. The comparison of the recovered result and original result is shown in Fig. 5.24. The recovered results are in good agreement with the original one. Thus, it proves that the damage load predicted by ML can be used to recover the crash acting in interval of the training points.

Crash collision points are outside of training hemisphere quarter

For a large-scale engineering practice, it is impossible to have training data covering everywhere of the shell structure. In this study, the training data are only located in a quarter of the hemisphere shell. Thus, in principle we may not be able to use the training data to intelligently "interpolate" collision loading conditions outside the training quarter. However, since the hemisphere shell is symmetric, we may only need the training data in a symmetric portion of the shell structure, and predict damage load conditions in everywhere of the shell. In this case study, we will illustrate how to utilize the symmetric property of the hemisphere shell structure to predict the load conditions when the collision occurred outside of the training area. Since the present training data are obtained based on a finite element mesh, which has a symmetric periodicity of every 90° in circumference rotational direction as shown in Fig. 5.25.

In these cases, the collision occurred at longitude $180°$ and latitude $55°$, as shown in Fig. 5.26. The information of the load is listed in Table.5.21. First of all, we equally divide the hemisphere into four quarters as shown in Fig. 5.25. Each quarter has 1610 nodal points. Then we transfer



(a) (b)

Figure 5.24: Stress distribution of Dynamics case 4: (a) stress distribution of test data; (b) recovered stress distribution generated by ML results.

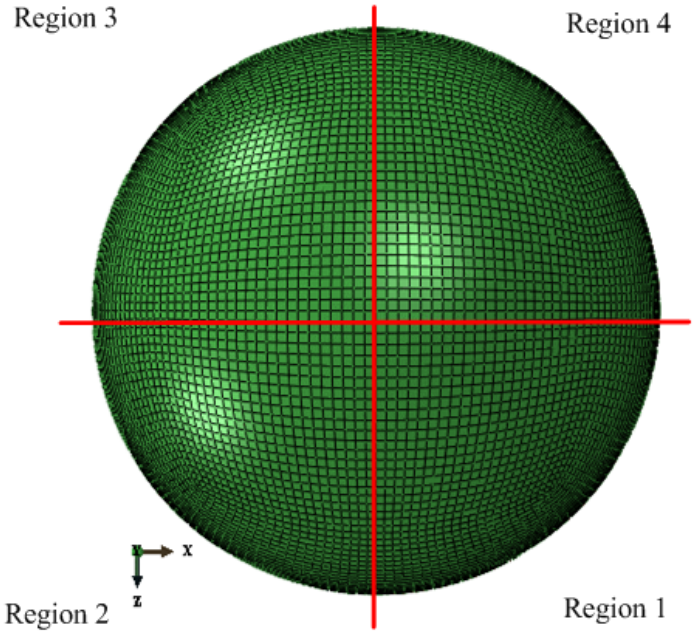


Figure 5.25: Symmetry properties of hemisphere shell and its finite element mesh

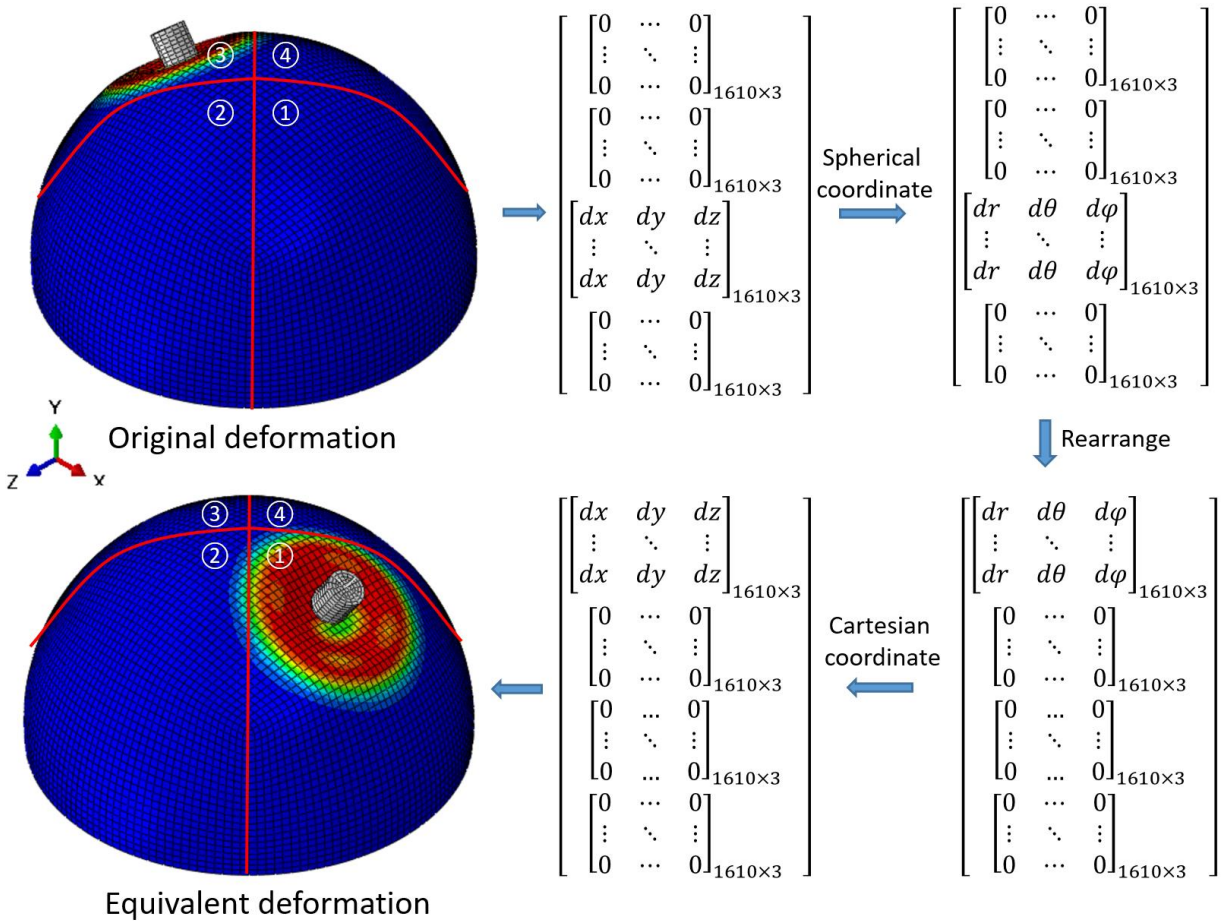


Figure 5.26: Deformation transferring process

all of the training data from the first quarter to the fourth quarter based on its four-fold periodicity. Since the training area is in the first quarter of the shell, all effective values are in the first quarter of the deformation matrix. Then, we trained the neural network with these processed data following the flowchart.

In order to let the neural network identify the testing data, we should also rearrange the testing data as the training data. The rearrange process is shown in Fig.5.26. Then we input the equivalent deformation matrix to the trained neural network by following the flowchart in Section 2 and get the predicted result. The predicted results will be in the first quarter of the shell because of the transformation. Finally, we turn the predicted coordinates back to its original quarter. The predicted results of the testing case is shown in 5.21.

The numerical simulation results show that the proposed method can successfully predict location of the load with an error less than 10%. The predicted duration is relatively larger but is still

Table 5.21: Predicted results of the impact on $(180°, 55°)$

	Testing value	Predicted value	errors
Location x	-0.577	-0.615	6.586%
Location y	0.817	0.773	5.336%
Location z	0	0.026	2.6%
Normal speed (m/s)	10.5	10.04	4.756%
Tangential speed (m/s)	10.5	9.56	8.706%
Duration (s)	0.01	0.01193	19.302%

in the right interval between 0.01s and 0.015s. According to the predicted interval, we can reduce our target area and carry out further refined prediction step by step until we get the accurate result. Therefore, by use of the symmetric property, neural network can also predict the collision load conditions that occurred outside of the training area.

Chapter 6

Simulation and Results of Car Collisions

This chapter continues to develop a novel deep learning computational framework to determine and identify the damage load conditions for cars.

6.1 Data Collection

Finite Element Model and Boundary Conditions

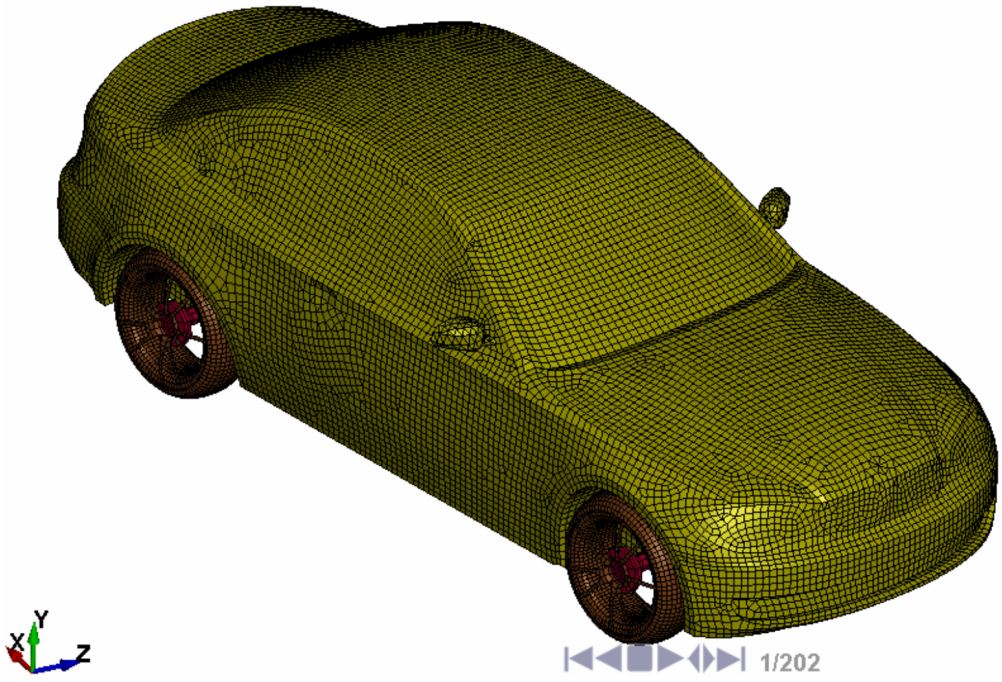


Figure 6.1: FEM model of a car

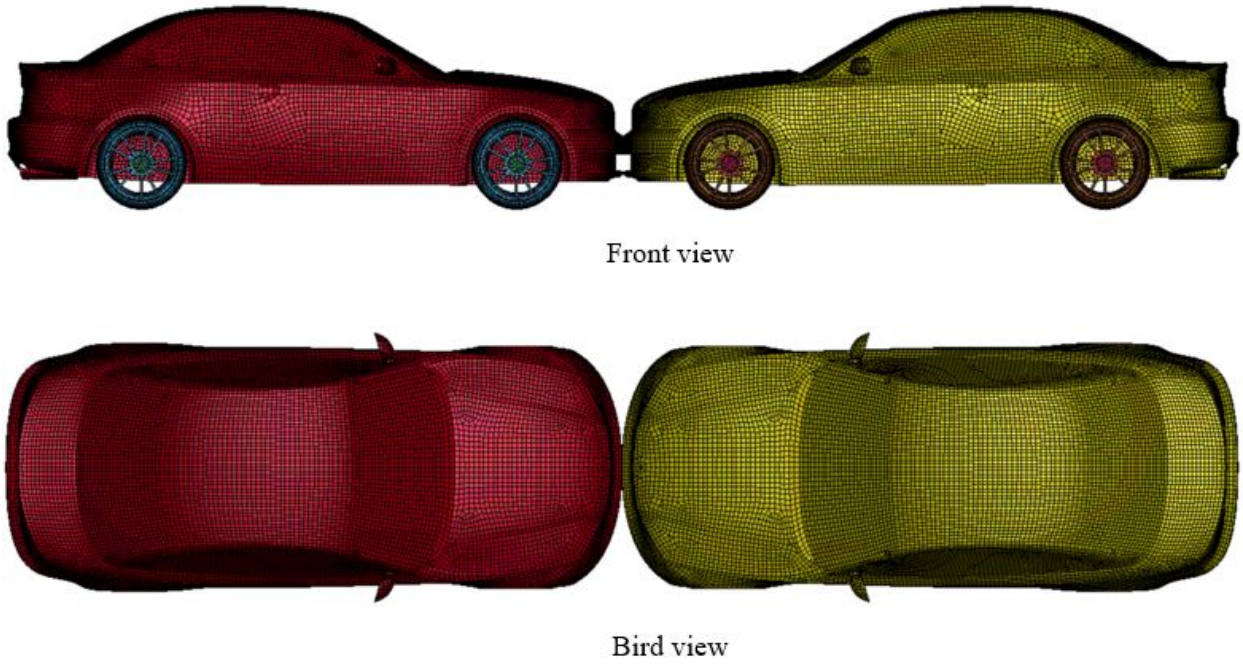


Figure 6.2: FEM models of two cars

Table 6.1: Geometry information of the model

Features	Values
Length	4.082 m
Width	1.783 m
Height	1.192 m
Wheel base	2.492 m
Car body shell thickness	3.5 mm

Based on the geometry of a real BMW sedan, the 3D car model was generated and meshed by use of Solid3D. The geometry information of the car is shown in Table 6.1. The 3D model is shown in Fig. 6.1. A car model contains three main parts which are the car body shell, the axils and the wheels. Then two cars were generated and positioned head to head to mimic the crashing position, as shown in Fig. 6.2.

Then the geometric model was imputed to the professional car crash simulation software LS-DYNA. All of the simulations in this study were carried out by used of LS-DYNA [88]. (LS-DYNA is developed by the Livermore Software Technology Corporation (LSTC). It is a general-purpose finite element program capable of simulating complex real-world problems. It is used by

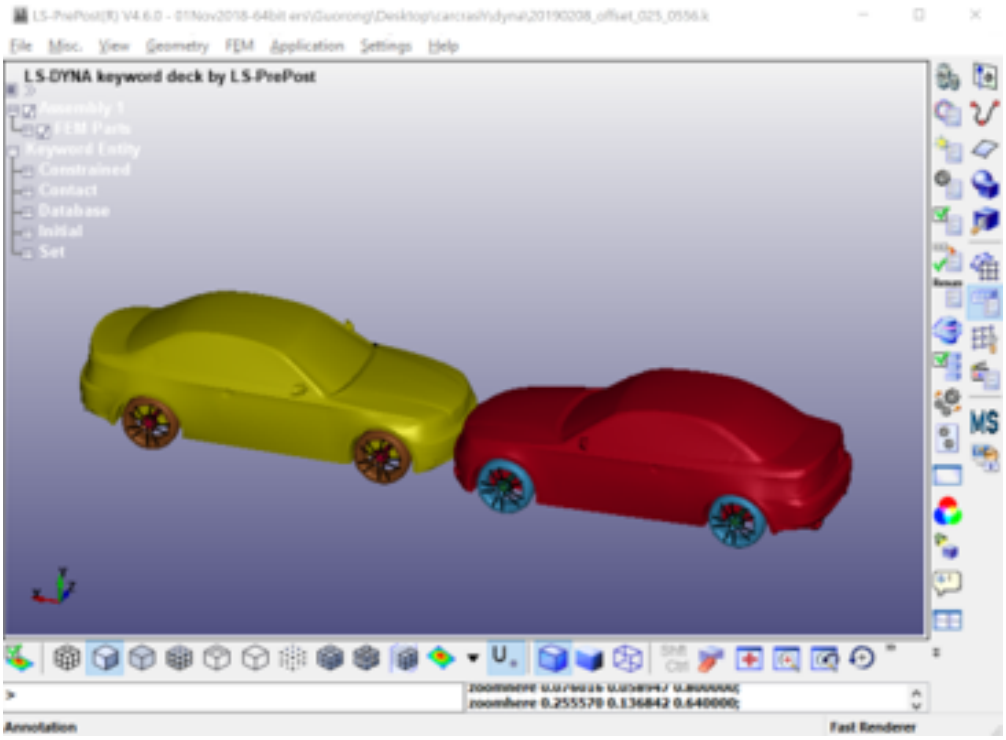


Figure 6.3: FEM models of two cars in LS-DYNA software interface

the automobile, aerospace, construction, military, manufacturing, and bioengineering industries.) Since the thickness of car body is much smaller than the length and the width of the car, the quadrilateral and triangle Belytschko-Tsay shell elements were adopted to model the car body and the wheels. The axles, which are two solid bars, were modeled by eight-node linear solid elements. The influence of different mesh sizes was investigated in a preliminary study. Finally, an element size of around $2cm$ was used in the model, which leads to totally 39194 elements and 39919 nodal points in one car. Since the rotation of the wheel and the axles had no impact on the crash results, rigid connections between the wheels and the axles and between the axles and the car body were adopted. A nodal point in the button of each four wheels was fixed on vertical direction but free on horizontal directions. Thus, the cars can move and rotate in X-Z plane but not on Y direction. Except those four nodal points, the degree of freedom of all other nodal points have no constrains. Thus, the nodal points can deform on any direction. The global coordinate origin was set in the center point between two cars. The contact relationship between two cars was modeled by the Automatic_Surface_To_Surface module of LS-DYNA. The static and dynamic coefficients of friction between all parts were set to 0.15. Hourglass energy, Stonewall, sliding interface and Rayleigh energy dissipations were computed and included in the energy balance. The models are shown in Fig. 6.3.

Material Properties

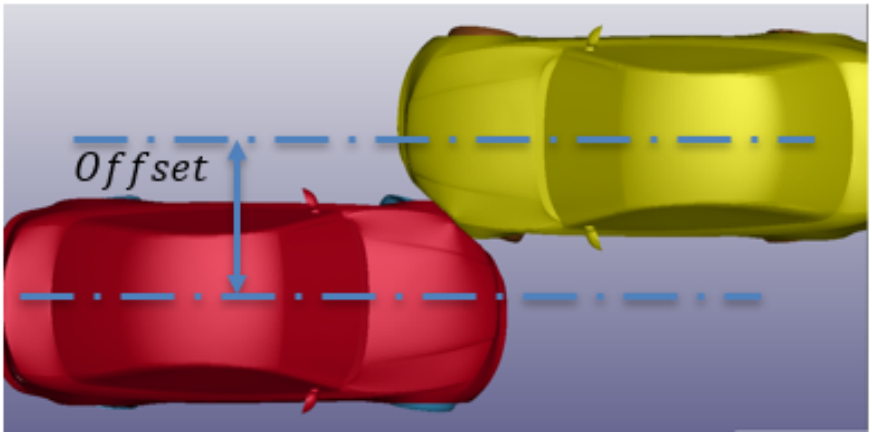


Figure 6.4: Offset of two crashed cars

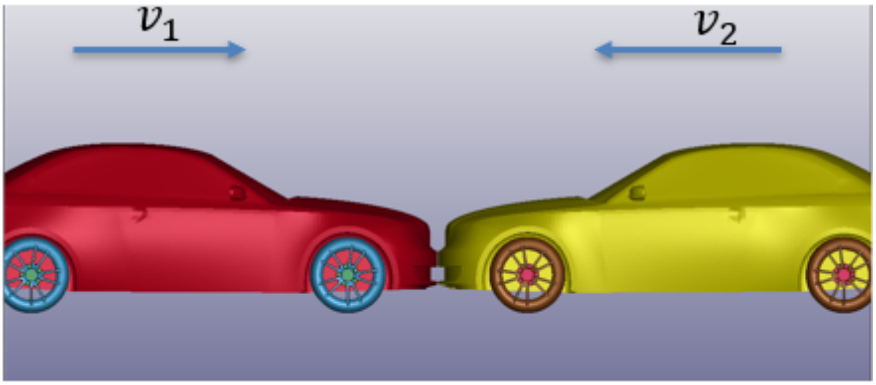


Figure 6.5: Velocities of two crashed cars

Models are developed in LS-DYNA and based on different parameters, the residual displacements of all the mesh points are calculated. There are three parameters, offset of the two cars, velocities of the two cars and the angle between two cars, which are shown in Fig. 6.4, Fig. 6.5, and Fig. 6.6.

According to the literature review of car industry, the SPCEN cold rolled steel plate, which is commonly used in auto body parts, was chosen as the body material of the car model [68]. The thixocast A356 alloy which is also commonly used on car wheels, was adopted as the wheel material [114]. The strain-stress curves of the two materials are shown in Fig. 6.7, and Fig. 6.8.

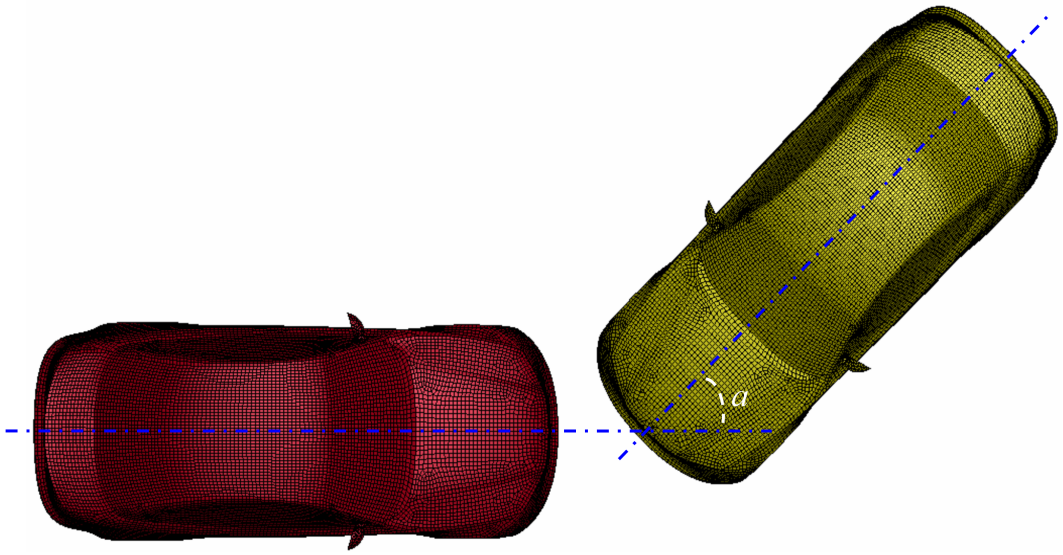


Figure 6.6: Angle between two crashed cars

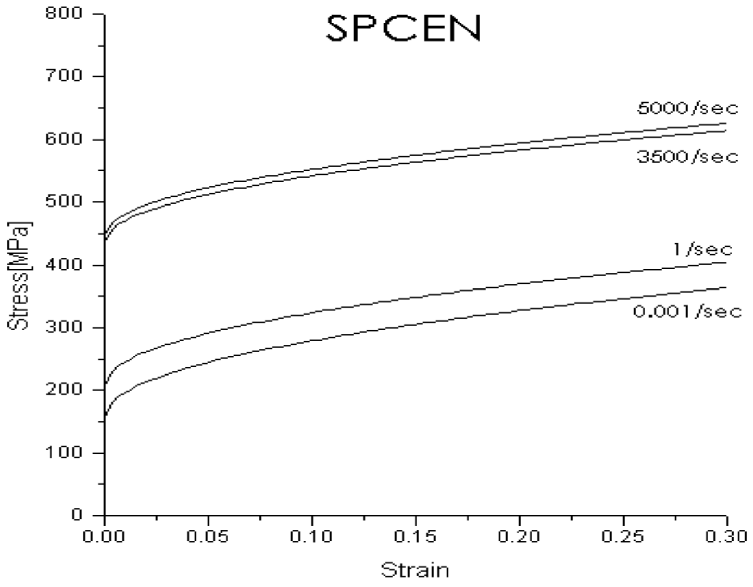


Figure 6.7: Strain-stress curve of the SPCEN steel

The strain and stress curve of the SPCEN and A356 alloy follows the Johnson-Cook constitutive model. The equation of the Johnson-Cook model is as follows:

$$\sigma = [A + B(\epsilon^p)^n][1 + Cln(\frac{\dot{\epsilon}^p}{\dot{\epsilon}_0})][1 - (\frac{T - T_r}{T_m - T_r})^n] \tag{6.1}$$

where A, B, C, m, n are material constants; ϵ^p is the plastic strain; $\dot{\epsilon}^p$ is the strain rate; $\dot{\epsilon}_0$ is the

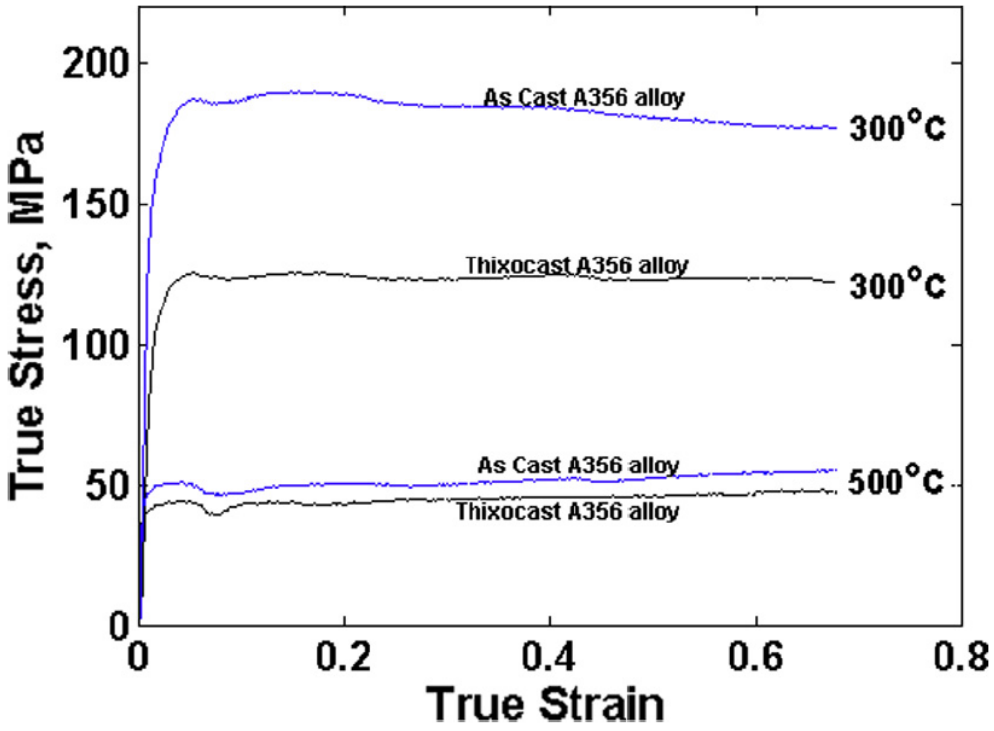


Figure 6.8: Strain-stress curve of the thixocast A356 alloy

Table 6.2: Johnson-Cook model parameters

Model Part	A(MPa)	B(MPa)	n	C	m	Tm	Tr	E(Gpa)	poisson
Car body	208	350	0.48	0.14	0.31	1093.15	298.15	210	0.3
Wheels	114.64	35.56	0.0243	0.7469	0.0707	973.15	298.15	72.4	0.33

reference strain rate; T is the absolute temperature; T_m is the melting temperature and T_r is the reference temperature.

The values of the Johnson-Cook model parameters adopted in this model are listed in Table 6.2.

Cases Design

To some extent, the variety, quantity and quality of the data determine the performance of the DNN model. To simulate various crash positions which would happen on the road, we offset the initial position of the cars transversely with different center line distances and rotate the cars with different crash angles, as shown as in Fig. 6.4 and Fig. 6.5. The values of the offset distances are set

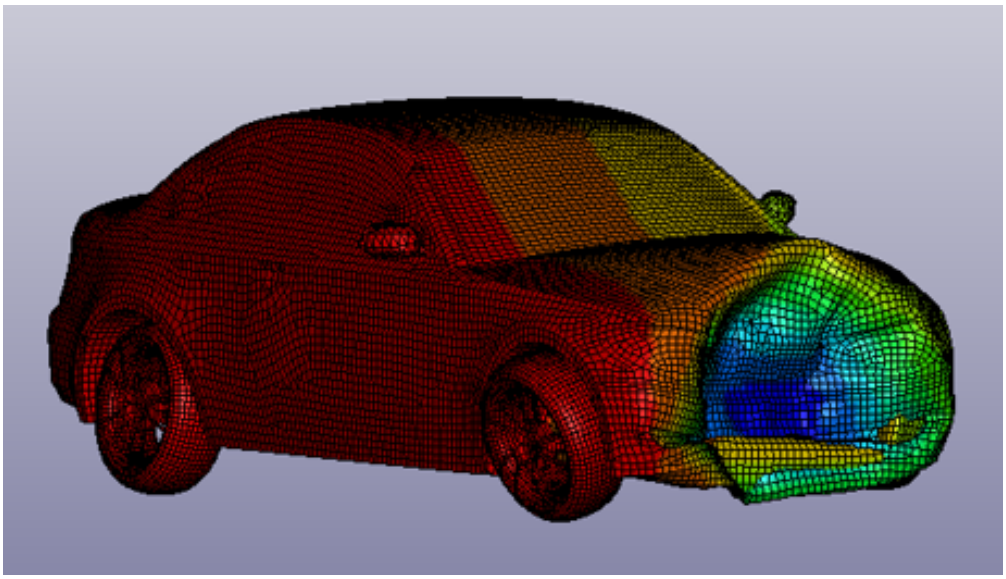


Figure 6.9: Residual displacement of a crash car after collision

Table 6.3: Initial speed combinations (km/h)

Car 2 \ Car 1	10	20	30	40	50	60	70
0	10-0	20-0	30-0	40-0	50-0	60-0	70-0
10	10-10	20-10	30-10	40-10	50-10	60-10	70-10
20		20-20	30-20	40-20	50-20	60-20	70-20
30			30-30	40-30	50-30	60-30	70-30
40				40-40	50-40	60-40	70-40
50					50-50	60-50	70-50
60						60-60	70-60
70							70-70

to $0m$, $0.25m$, $0.5m$, $0.75m$, $1.0m$, $1.25m$ and $1.5m$. The values of angles are set to $15°$, $30°$ and $45°$. Also, each car was initialized with seven different crash speeds which are $10km/h$, $20km/h$, $30km/h$, $40km/h$, $50km/h$, $60km/h$ and $70km/h$. The combinations of the car speeds are listed in Table6.3. To prevent the DNN model to make decision due to the initial difference between the cars instead of the plastic deformation, the FEM models of two cars were set exactly the same to each other. Since the FEM model of two cars are exactly the same, the speed combination $20-70km/h$ and $70-20km/h$ were treated as one case. Therefore, combining the different offset distances, angles and speeds one by one, there were totally 320 crashing cases were generated. The simulation began at the final stage before crashing when the distance between two cars was $2mm$, and ended in $0.2s$ after crash when the cars were completely separated. For each case, we have

$79,000$ mesh points and each point has residual displacements along three different directions. Thus, the input of the neural network is around $320 * 79,000 * 3$. The residual displacement is shown in Fig. 6.9.

6.2 Animation Results

Fig. 6.10 shows one case of animation of car collisions as time goes on with zero angle while Fig. 6.12 shows one case of animation of car collisions as time goes on with non-zero angles. The time ranges from $0.04s$ to $0.2s$. During the first $0.04s$ period, the two cars are running at their own speeds without any contact or collision.

In Fig. 6.10, velocities of left and right car are both $70km/h$ and the offset is $0.5m$. Fig. 6.11 shows the strain of the left car after car collision in the zero-angle case. From the figure, there is serious damage to the front of the vehicle while the rear of the car only experiences minor or no damage. From the deformation extracted from LS-DYNA, the hitting point tends to have larger deformation than other points, which match the phenomenon happen in real life. The center of the damage region is more to the reader's right direction because of the offset influence.

In Fig. 6.12, velocities of left and right car are both $70km/h$, the offset is $1.5m$ and rotation angle is $45°$. Fig. 6.13 shows the strain of the left car after car collision in the non-zero angle case. From the figure, there is serious damage to the front of the vehicle while the rear of the car only experiences minor or no damage. The hitting point tends to have larger deformation than other points, which match the phenomenon happen in real life. In addition, from the deformation information, the direction of the impact loads is not parallel to the axis of the car, which is because of the oblique impact instead of direct impact. Also, there exits the rotation of tires.

6.3 Discussions on Metrics to Evaluate Machine Learning Algorithms

Discussions on Cross-validation

The first machine model we tried had one hidden layer with 256 neural. However, the total loss during the training process could not decrease too much, which indicated the underfitting phenomenon. Usually people use one hidden layer for simple tasks, but nowadays research in deep neural network architectures show that many hidden layers can be fruitful for difficult object [8, 24, 111]. Based on the experience and previous research, a new machine learning model with three hidden layers and the sizes of $256, 64, 8$ is built up. We use 10-fold cross-validation to calculate the error of the prediction. The average error of offset is 1.58 %, the average error of velocities is 2.30 %, and the average error of angle is 3.23 %. The three error are relatively low indicates that

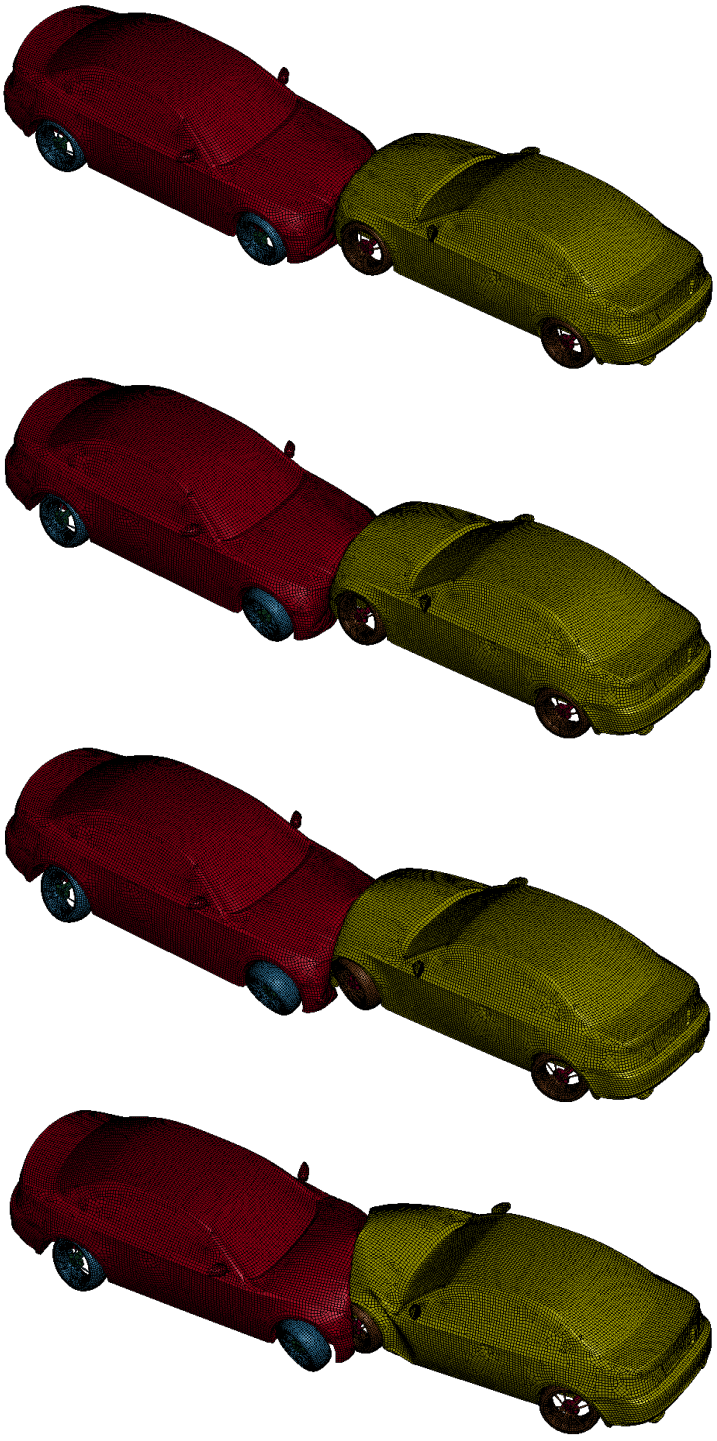


Figure 6.10: Animation of car collisions with zero angle (time: $0.04s$ - $0.2s$)

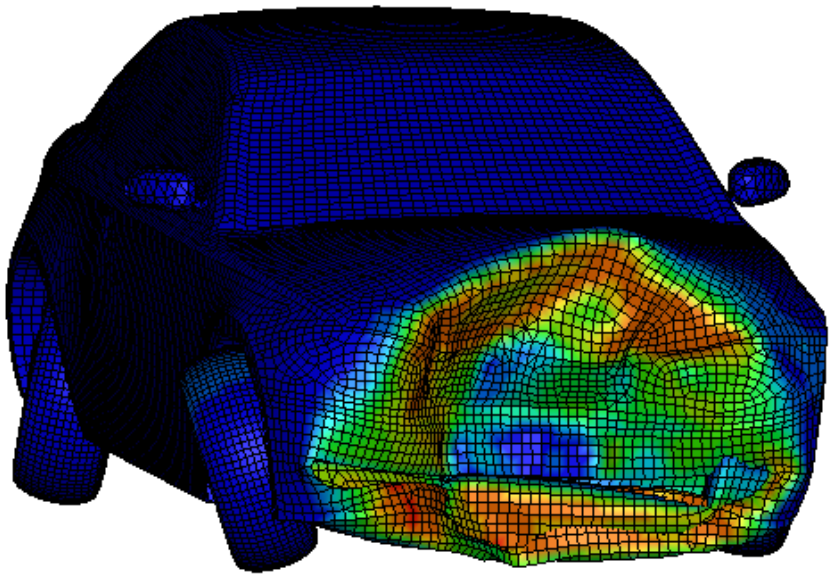


Figure 6.11: Strain of the left car after car collision with zero angle

Table 6.4: Results of offset

Testing cases	1	2	3	4	5
True Value (cm)	50	100	50	100	100
Predict Value (cm)	51.411	100.615	51.411	99.507	97.642
Error (%)	2.823	0.615	2.823	0.492	2.357

there is no overfitting in the process.

After training the selected neural network model with all the data, we get the finalized model. 50 different sets (around 15 % of size of training data) of test data which is not included in the training data are given to the finalized model to predict. Part of the results are shown in Table 6.4 (result of offset), Table 6.5 (result of velocities) and Table 6.6 (result of angles).

Modified L^2 Norm

The left animation in Fig. 6.14 shows the simulation of car collisions as time goes on with zero angle. The time ranges from $0.04s$ to $0.2s$. The velocities of left and right car are $75km/h$ and $25km/h$ and the offset is $0.5m$. We build up the model, let's call it the original model, in LS-DYNA based on the given parameters and perform the simulations. Then the residual displace-

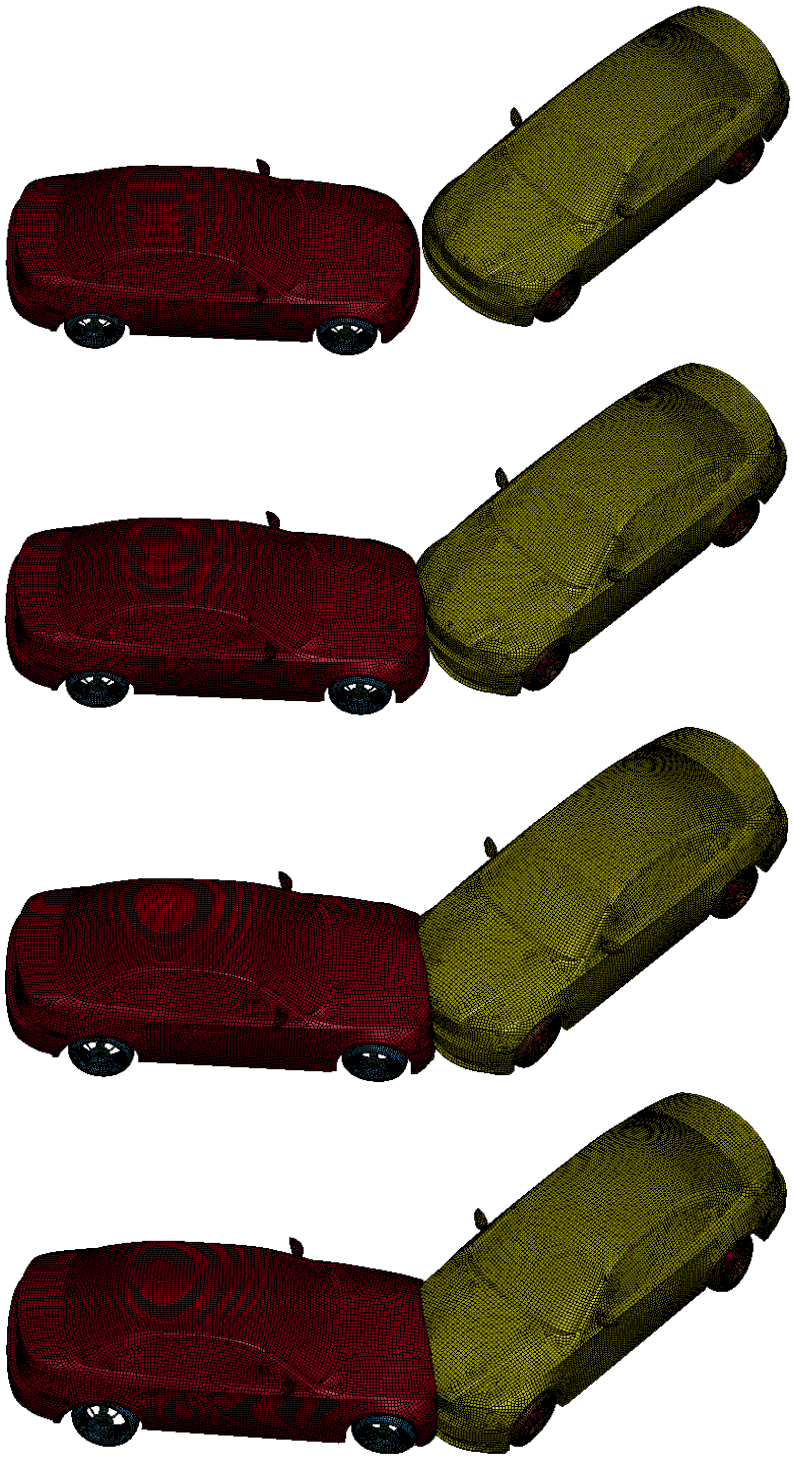


Figure 6.12: Animation of car collisions with non-zero angle (time: $0.04s$ - $0.2s$)

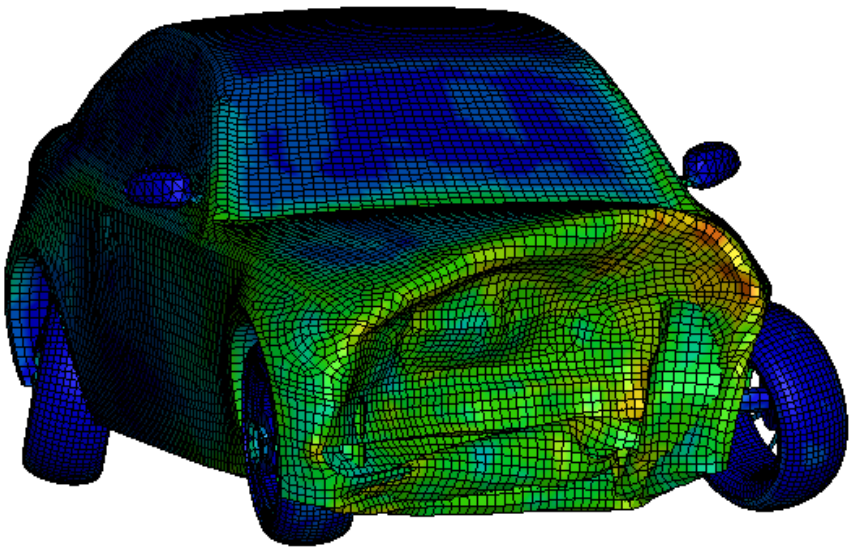


Figure 6.13: Strain of the left car after car collision with non-zero angle

Table 6.5: Results of velocities

Testing cases	1	0.5m	2	0.5m	3	1m	4	1m	5	1m
True Value (km/h)	35	15	75	25	35	15	55	45	75	15
Predict Value (km/h)	35.15	14.97	74.14	25.04	34.38	14.927	55.15	44.43	74.70	14.92
Error (%)	0.45	0.18	1.14	0.17	1.76	0.48	0.28	1.26	0.39	0.48

ment is extracted from LS-DYNA software. After that, machine learning neural network model predicts the offset, velocities and angles based on the residual displacements of the two cars. The predicted values are $0.50m$ offset, $74.14km/h$ and $25.04km/h$ velocities and 0 angles. Based on the predicted parameters, a new model is build up in LS-DYNA to perform the same simulation, let's call it the predicted model. In this new model, we use the same materials parameters and mechanical methods that are used in the original model. The right animation in Fig. 6.14 shows the simulation of car collisions of the predicted model as time goes on.

Fig. 6.15 shows the comparison of the residual displacement, including x and y directions, of the left car between the original case and predicted case after car collisions. The left two plots are from the original model while the right two plots are from the predicted model. Fig. 6.16 shows the comparison of the residual displacement, including x and y directions, of the right car between

Table 6.6: Results of angles

Testing cases	1	2	3	4	5	6	7
True Value (degree)	0	15	20	30	35	45	55
Predict Value (degree)	0.032	14.934	20.343	29.698	35.451	44.982	54.282
Error (%)	null	0.440	1.715	1.007	1.286	0.040	1.305

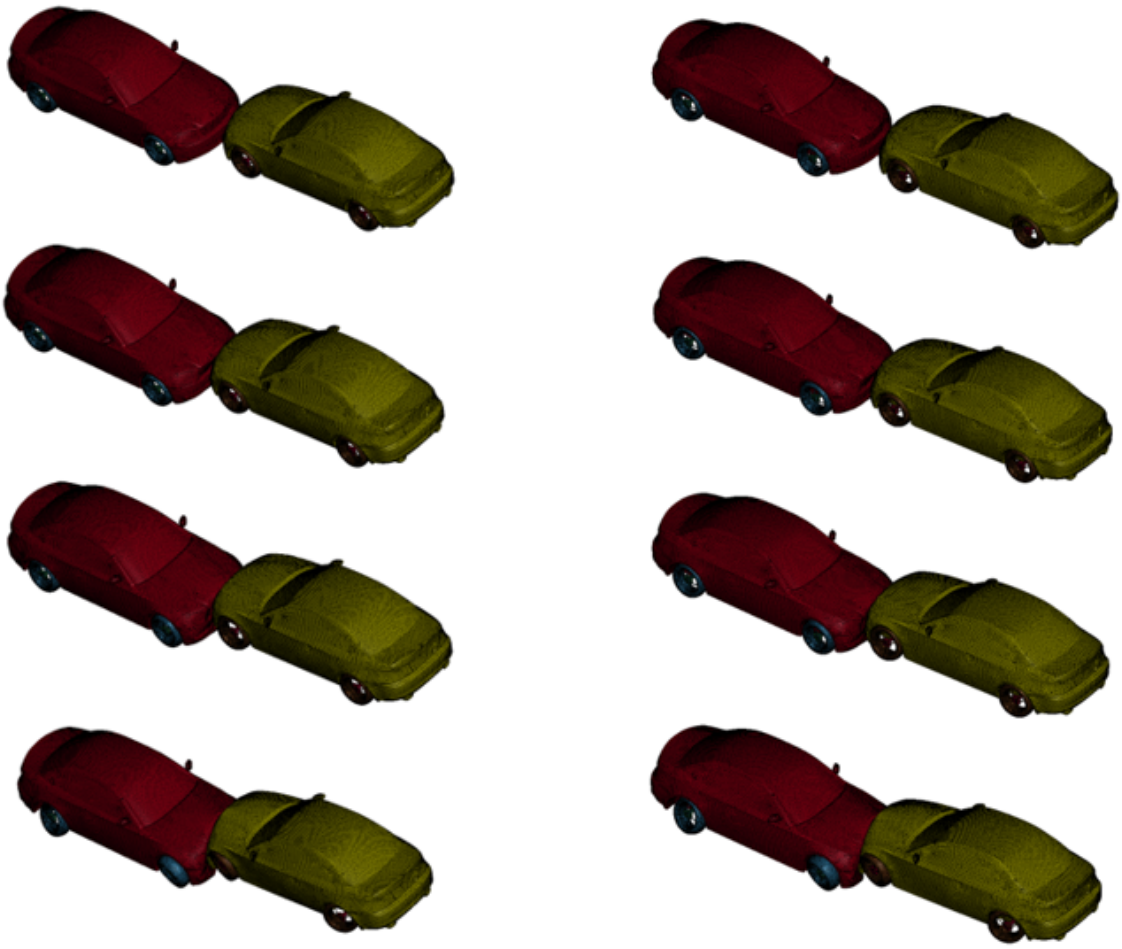


Figure 6.14: Animation comparison between the original case and predicted case after car collisions

the original case and predicted case after car collisions. The left two plots are from the original model while the right two plots are from the predicted model.

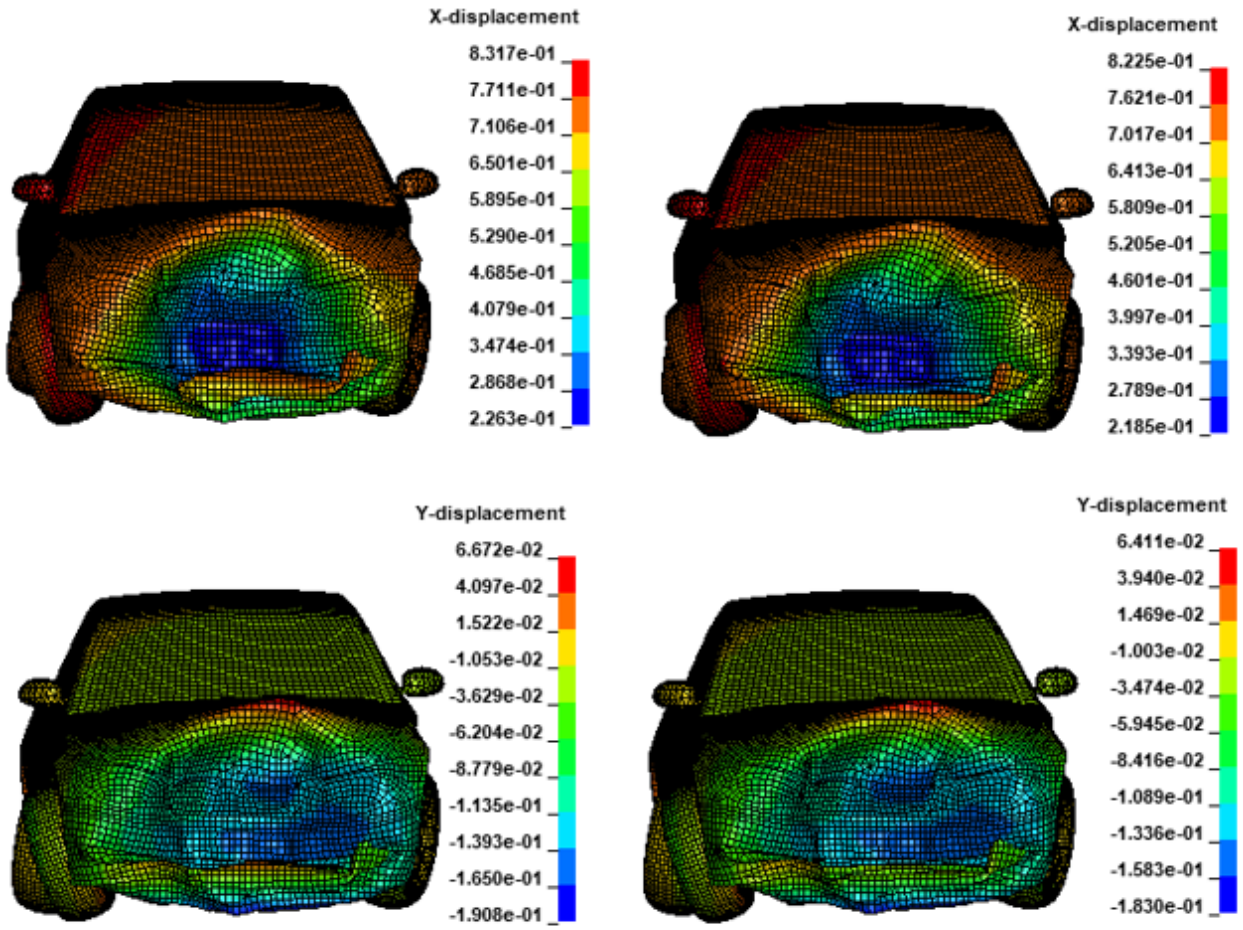


Figure 6.15: Comparison of the residual displacement of the left car between the original case and predicted case after car collisions

From Fig. 6.14, Fig. 6.15 and Fig. 6.16, we can tell that the prediction of our machine learning model is very promising. To numerically measure the difference, we use modified L^2 norm. The detailed calculation method is explained in Chapter 4. The modified L^2 norm is 1.78 % for the left car and 3.64 % for the right car. From these two numbers, we can tell the predicated model is very similar to the original model, which indicates a good prediction ability of the machine learning neural network.

By stochastically selection of original models, predicted models are set up afterwards. Then modified L^2 norm is applied to tell the quality of the given machine learning model. The modified L^2 norm is a new metric we bring up to estimate the goodness of a neural network. It is a pro-

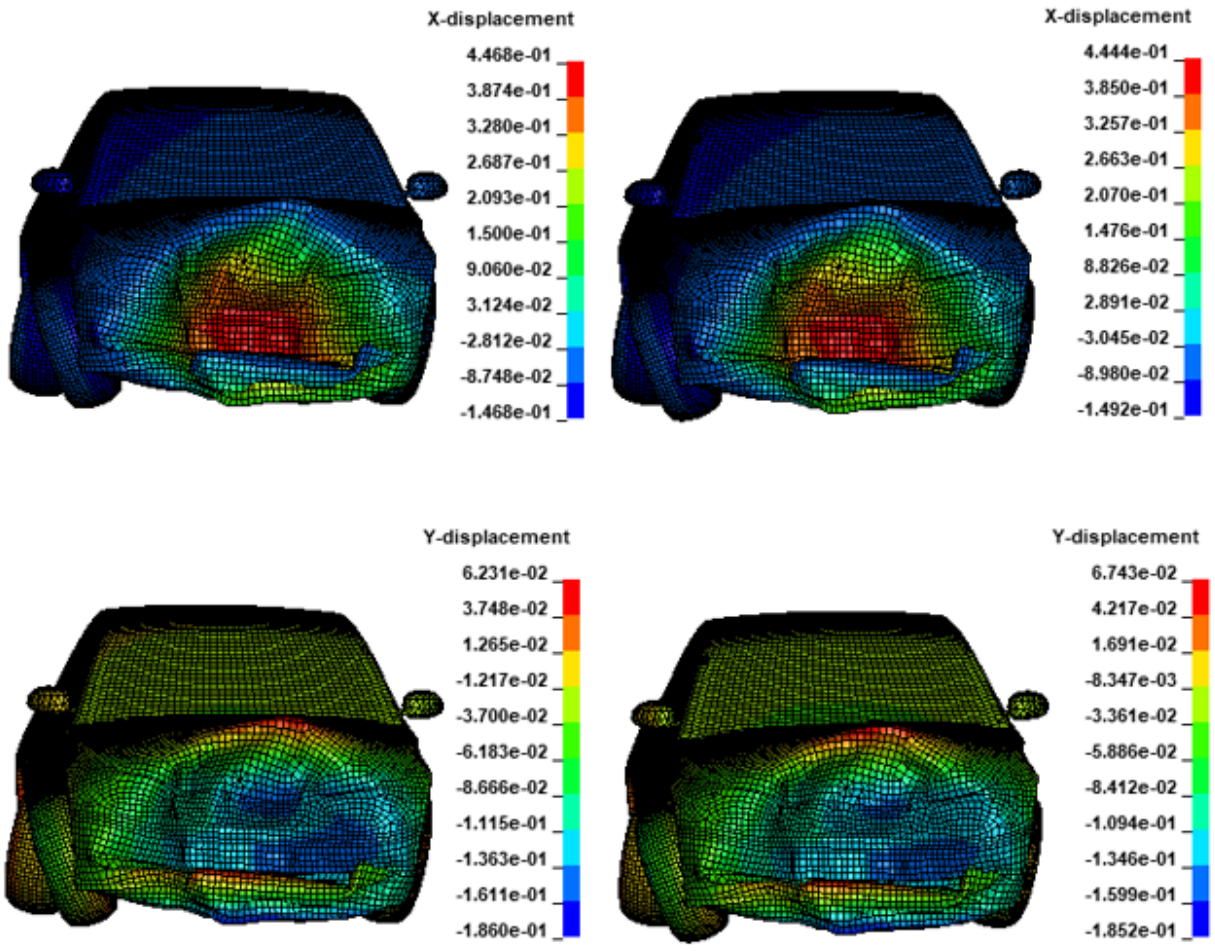


Figure 6.16: Comparison of the residual displacement of the right car between the original case and predicted case after car collisions

fessional term that is widely use in computational mechanics nowadays, and could also be used in machine learning algorithm. It brings more explanatory to the neural network, the so called black box algorithm.

6.4 Discussions on Activation Functions

In this discussion, we use offset prediction as an example. Using the same machine learning model with three hidden layers and the same training data, different activation functions are applied to train the model to predict offset. The results are shown in Table 6.7. Among these five activation functions, ReLU is the best activation function in terms of error. The average error of

Table 6.7: Comparison of different activation functions

Activation Function	Error (%)	Training Time (seconds)	Total Loss
ReLU	1.28	2824.50	48.68
Sigmoid	4.029	2932.06	21.81
Tanh	4.062	2848.56	21.56
Softmax	4.77	3182.80	203.52
ReLU Square	4.398	2854.70	219.47

offset is 1.28 %, while the other methods all get around 4 % error. The same phenomenon happens during the process to predict velocities and angles.

Besides, ReLU takes relatively less time than other methods. Finishing a 10-fold cross validation simulation, ReLU model uses around 2824.50 seconds for a 10-core computer, while it takes around 2392.06 seconds for sigmoid, 2392.06 seconds for softmax method, 2392.06 seconds for softmax and 2392.06 seconds for ReLU Square. It is because the other methods takes more time to calculate the gradient during the backpropagation process, especially it becomes a very complex equation to calculate the derivative of softmax method, while ReLU's gradient is very easy to calculate, as shown in the following equation:

$$f'(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases} \tag{6.2}$$

The derivative of these five methods are shown in Fig. 6.17, Fig. 6.18, Fig. 6.19 and Fig. 6.20. The derivative of softmax function depends on the other parameters and input, not just the x. From these four figures, we can see clearly that ReLU has a simpler gradient. The biggest advantage of ReLU is indeed non-saturation of its gradient, which greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid and tanh functions. But it is not the only advantage. Another nice property is that compared to tanh / sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

In addition, one interesting finding is that the method with the smallest total loss doesn't always lead to the least error, such as sigmoid method. It is an obvious symptom indicating overfitting that training error is very small while the test error is large. In addition, the largest total loss doesn't lead to the least error either, such as softmax and ReLU Square method. It may be because of underfitting that the both the training and test error are both large. ReLU's training error is relatively small, which leads to the smallest test error.

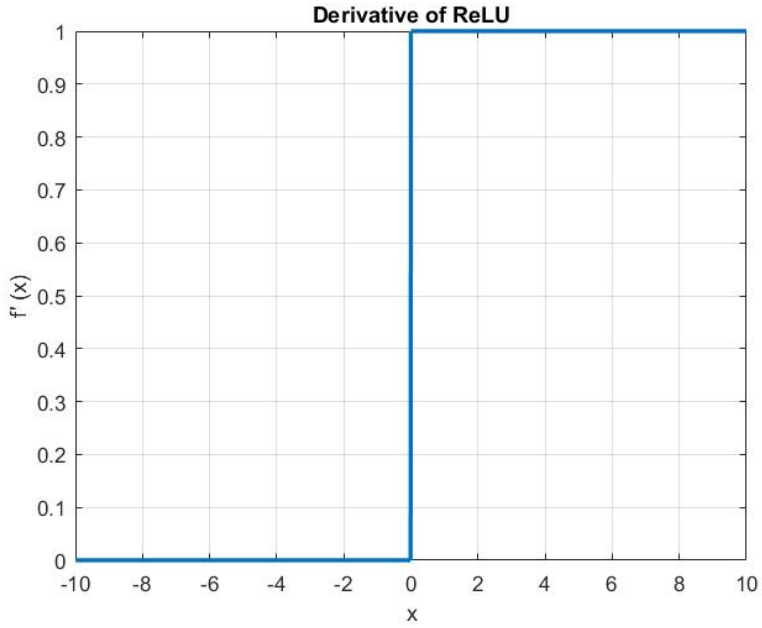


Figure 6.17: The derivative of Relu function plot

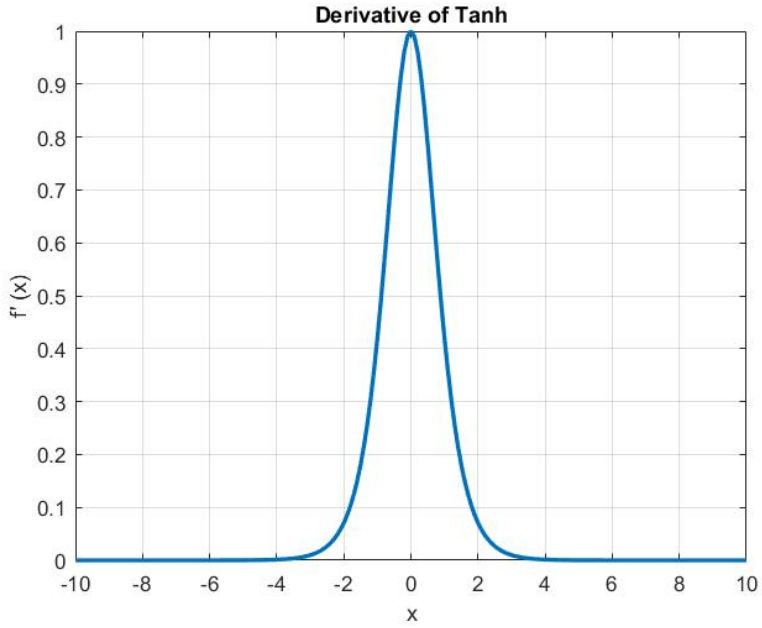


Figure 6.18: The derivative of Tanh function plot

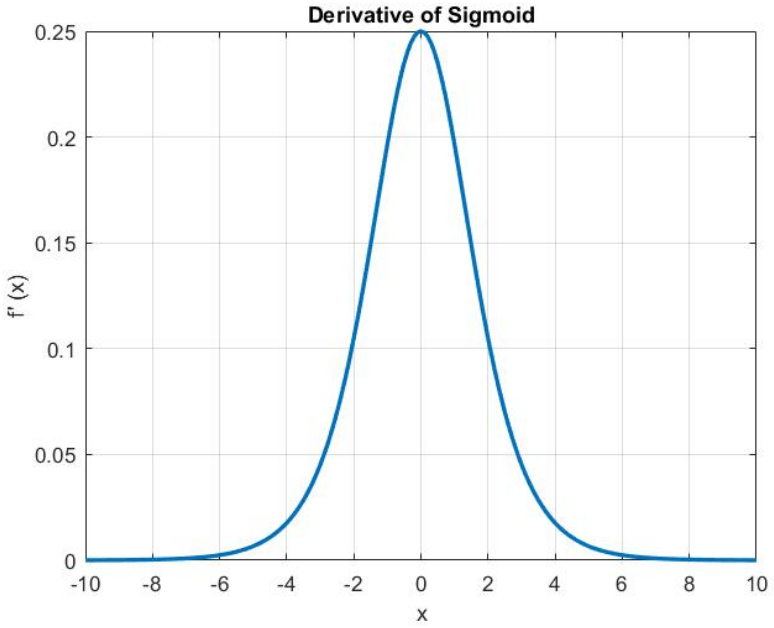


Figure 6.19: The derivative of Sigmoid function plot

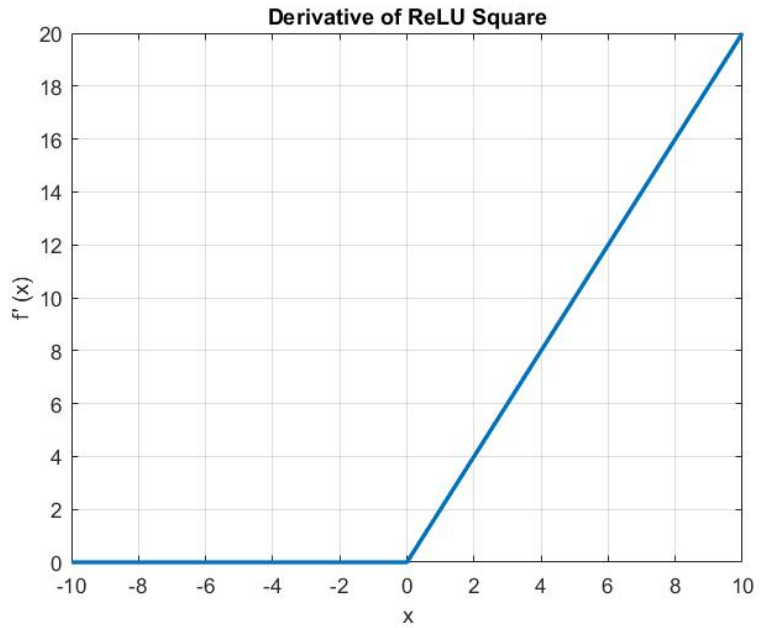


Figure 6.20: The derivative of Relu Square function plot

Table 6.8: Comparison of different cost functions

Cost Function	Error (%)	Training Time (seconds)	Total Loss
Maximum Loss	1.51	2833.19	204.21
Mean Loss	1.35	2846.21	49.49

6.5 Discussions on Cost Functions

We use offset prediction as an example. Using the same machine learning model with three hidden layers and the same training data, different cost functions are applied to train the model to predict offset. The results are shown in Table 6.8. Among these two activation functions, there are no obvious differences between maximum loss method and mean loss method in this study. However, the total loss has a large difference since they have the totally different way to calculate the loss. The same phenomenon happens during the process to predict velocities and angles.

6.6 Discussions on Data Filtering

Without data filtering, it is still possible to predict offset, velocity and angles separately or at the same time. A same machine learning model with three hidden layers and sizes of $256, 64, 8$ is developed to predict offset, velocity and angles at the same time to provide a comparative test. The same training data is used. After 10-fold cross-validation, the average errors of the prediction are 3.49 %, 15.25 % and 11.30 % separately for the three parameters.

After using data filtering technology, the average error of offset is 1.58 %, the average error of velocities is 2.30 %, and the average error of angle is 3.23 %. The accuracy of offset, velocity and angles get improved by around 2 %, 13 % and 8 %.

From the comparison, we can tell that it decreases the error a lot by dividing one neural network to several small ones and predict output step by step. Between each step, data filtering technology is necessary to avoid un-necessary data and noise.

Chapter 7

Closing

7.1 Summary

In the simulation of cantilever beams with inelastic materials, we applied deep (machine) learning techniques to solve an inverse engineering problem of identification of the location, amplitude, and duration of impact forces, both in static and dynamic conditions, based on the permanent residual deformation as well as the permanent strain distributions. For static problems, the developed machine learning algorithm can predict both the location as well as the amplitude of the force with high accuracy. The prediction on the location of the applied load is not necessarily from the training data. After studying the training data, the machine learning algorithm can automatically use interpolation to find the applied load location that is not in the training data. This is to say that we only need to train the neural network with a limited number of loading sets, it can then predict any loading locations on the boundary of the cantilever beam. This is a remarkable success for an inverse problem solution that was impossible to realize because of the non-unique solutions for inverse problems. Based on this study, we have come to a conclusion that by using the forensic signatures such as permanent deformation and residual plastic strain distributions one can uniquely (practically) determine the applied load conditions inversely with high accuracy for most engineering purposes.

For dynamic loading problems, the current version of our machine learning algorithms can also predict both the location and amplitude of applied loads with high accuracy, whereas the accuracy of the prediction on the duration of the loads is not high, even though it still has good accuracy i.e. the error is within 5% to 10 %. These results are very much promising for both static and dynamic inverse problems. Results herein demonstrate that the ML or AI based approaches can solve many previously intractable problems.

In the simulation of elasto-plastic shell structures, we have developed an artificial intelligence based machine learning method to predict impact loading conditions during the collision of shell structures with rigid body impactor. In particular, we have applied the deep learning techniques to

solve a 3-D inverse engineering problem of identification of the location, amplitude, and duration of collision between an elastoplastic hemispherical steel shell and a rigid impactor, based on the permanent plastic deformation of the shell. For static problems, the developed machine learning algorithm can predict both the location as well as the amplitude of the collision force with high accuracy. This is a remarkable success for a three-dimensional inverse problem solution. Based on this study, we have come to a conclusion that by using permanent plastic deformation distribution as structural "forensic signatures" one can determine the damage load conditions reversely with a high accuracy for most engineering purposes.

For dynamic loading problems, the current version of our machine learning algorithms can also predict both the location and amplitude of applied loads with high accuracy, where the accuracy of the prediction on the collision duration is not high, even though it still has good accuracy. Also, by using symmetric property of the shell, we do not need to generate all the training data to cover every part of the shell, which can save many data storage as well as much computational time. These results presented in this paper are very much promising for both static and dynamic inverse problems, and they have demonstrated that the AI based machine learning approaches can solve some previously intractable problems. This is because the deep-learning algorithm developed here can not only fit the training data to achieve an optimum regression-based forecasting or prediction, but also interpolate the training data in a very high dimension prediction space. Thus, it can provide accurate predictions in failure analysis.

Based on the two previous results, in the car collision simulation, we have shown that the developed machine learning algorithm can accurately and uniquely identify impact loading conditions on different types of structures in an inverse manner, based on the residual plastic strain, plastic deformation or residual displacement as forensic signatures, in computer science methods applied to computational mechanics area. The dissertation presented the detailed machine learning algorithm, data acquisition and learning processes, and validation/verification examples.

In the machine learning algorithm, feature selection is first used to select the feature which fits best for the study purpose and the output prediction. After that, data standardization is applied to bring data into a common format that allows for collaboration. Then we use principal component analysis to save computation time. Data filtering is introduced to reduce the final error. By this way, one big neural network is divided into three small ones to predict the output step by step. Drop-out is bought in to prevent overfitting. K-fold cross validation is to test the generalization of the machine learning model. We also come up with a method combined with mechanical term, L^2 norm, to serve as the metric of the machine learning model. Finally, we give a comparison between the animation of the original model and the predicted model, in addition with the discussions on different activation functions and cost functions at last.

This development may have significant impact on forensic material analysis and structure failure analysis, and it provides a powerful tool for material and structure forensic diagnosis, determination, and identification of damage loading conditions in accidental failure events, such as car

crashes and infrastructure or building structure collapses.

7.2 Future Work

Many opportunities exist for future research.

More Applications

Artificial neural networks, due to some of its wonderful properties have many applications:

1. Image Processing and Character recognition: Given ANNs ability to take in a lot of inputs, process them to infer hidden as well as complex, non-linear relationships, ANNs are playing a big role in image and character recognition. Character recognition like handwriting has lot of applications in fraud detection (e.g. bank fraud) and even national security assessments. Image recognition is an ever-growing field with widespread applications from facial recognition in social media, cancer detention in medicine to satellite imagery processing for agricultural and defense usage. The research on ANN now has paved the way for deep neural networks that forms the basis of deep learning and which has now opened up all the exciting and transformational innovations in computer vision, speech recognition and natural language processing [87].

2. Forecasting: Forecasting is required extensively in everyday business decisions (e.g. sales, financial allocation between products, capacity utilization), in economic and monetary policy, in finance and stock market. More often, forecasting problems are complex, for example, predicting stock prices is a complex problem with a lot of underlying factors (some known, some unseen). Traditional forecasting models throw up limitations in terms of taking into account these complex, non-linear relationships. ANNs, applied in the right way, can provide robust alternative, given its ability to model and extract unseen features and relationships. Also, unlike these traditional models, ANN doesnt impose any restriction on input and residual distributions.

ANNs are powerful models that have a wide range of applications and they have far-reaching applications across many different fields in medicine, security, banking/finance as well as government, agriculture and defense. In the future, we will explore more on the usage of ANN for computational mechanics.

Experiments

It is experiment that provides the evidence that grounds this knowledge. Experiment plays many roles in science. One of its important roles is to test theories and to provide the basis for scientific knowledge. We are able to predict the loading condition based on the deformations of cars in simulation process. It is still a problem to predict the loading condition based on the photos or real cars after car accidents and structures after earthquakes, which is a very promising project.

Experiments should be performed in the future to support the theory. In genetics, simulations can compress time. Also, simulations are most valuable, many believe, when teamed with actual experiments [128].

Data, Algorithm and Infrastructure

Today, data has become the new oil. It is arguably the worlds most valuable resource. While technologies such as smartphones and the internet have made data abundant and ubiquitous, those who succeed will be the ones who know how to leverage the data they have access to. However, in computational mechanics area, there is no easy access to real big data. Big Data means a large chunk of raw data that is collected, stored and analyzed through various means which can be utilized by organizations to increase their efficiency and take better decisions. Big Data is the combination of these three factors; High-volume, High-Velocity and High-Variety. The current data is far less than enough to build up a completed computational mechanics simulation system to perform projects. Better data always lead to better results.

Also, as data is increasing, some traditional algorithms can't handle such a huge database. The improvement of algorithms is essential in the future. In addition, the infrastructure of computational mechanics is not easily accessible. In general, they are either too expensive or not user-friendly to researchers nowadays. There still exists a lot of work to be done to build up a completed big data system, to improve the algorithms and to build the computational mechanics infrastructure.

7.3 Broader Impact of the Dissertation

Recent progress in artificial intelligence has renewed interest in building systems that learn and think like people. Many advances have come from using deep neural networks trained end-to-end in tasks such as object recognition, video games, and board games, achieving performance that equals or even beats humans in some respects. Advances in deep learning and other machine learning algorithms are currently causing a tectonic shift in the technology landscape. Technology behemoths like Google and Microsoft are engaged in an artificial intelligence arms race, gobbling up machine learning talent and startups at an alarming pace. They are building AI technology war chests in an effort to develop an insurmountable competitive advantage.

Machine learning is popular because computation is abundant and cheap. Abundant and cheap computation has driven the abundance of data we are collecting and the increase in capability of machine learning methods. Machine learning, reorganized as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics and probability theory. Our project is the novel application machine learning in traditional civil engineering, which may lead

the trend in the future.

In summary, this work is a novel application of computer science and computational mechanics. And we believe data-based machine learning applications will become the trend of the computational mechanics in the data era.

Bibliography

[1] Osama Abdeljaber et al. “Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks”. In: *Journal of Sound and Vibration* 388 (2017), pp. 154–170.

[2] Ken Aho, DeWayne Derryberry, and Teri Peterson. “Model selection for ecologists: the worldviews of AIC and BIC”. In: *Ecology* 95.3 (2014), pp. 631–636.

[3] Hirotugu Akaike. “Implications of informational point of view on the development of statistical science”. In: *Selected Papers of Hirotugu Akaike*. Springer, 1994, pp. 421–432.

[4] Jose E Andrade et al. “Multiscale modeling and characterization of granular matter: from grain kinematics to continuum mechanics”. In: *Journal of the Mechanics and Physics of Solids* 59.2 (2011), pp. 237–250.

[5] GQ Asrar et al. “Estimating absorbed photosynthetic radiation and leaf area index from spectral reflectance in wheat 1”. In: *Agronomy journal* 76.2 (1984), pp. 300–306.

[6] Iñigo Barandiaran. “The random subspace method for constructing decision forests”. In: *IEEE transactions on pattern analysis and machine intelligence* 20.8 (1998).

[7] Yoshua Bengio, Yann LeCun, et al. “Scaling learning algorithms towards AI”. In: *Large-scale kernel machines* 34.5 (2007), pp. 1–41.

[8] Yoshua Bengio et al. “Greedy layer-wise training of deep networks”. In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[9] Mairead L Bermingham et al. “Application of high-dimensional feature selection: evaluation for genomic prediction in man”. In: *Scientific reports* 5 (2015), p. 10312.

[10] Bastian Bohn et al. “Analysis of car crash simulation data with nonlinear machine learning methods”. In: *Procedia Computer Science* 18 (2013), pp. 621–630.

[11] Andrej Bratko et al. “Spam filtering using statistical data compression models”. In: *Journal of machine learning research* 7.Dec (2006), pp. 2673–2698.

[12] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[13] Carla E Brodley, Mark A Friedl, et al. “Identifying and eliminating mislabeled training instances”. In: *Proceedings of the National Conference on Artificial Intelligence*. 1996, pp. 799–805.

[14] Alex Castrounis. *Artificial Intelligence, Deep Learning, and Neural Networks Explained*. 2016.

[15] Gavin C Cawley and Nicola LC Talbot. "On over-fitting in model selection and subsequent selection bias in performance evaluation". In: *Journal of Machine Learning Research* 11.Jul (2010), pp. 2079–2107.

[16] Jinghui Chen et al. "Outlier detection with autoencoder ensembles". In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM. 2017, pp. 90–98.

[17] Chedsada Chinrungrueng and Aimamorn Suvichakorn. "Fast edge-preserving noise reduction for ultrasound images". In: *IEEE Transactions on Nuclear Science* 48.3 (2001), pp. 849–854.

[18] Miao Chong, Ajith Abraham, and Marcin Paprzycki. "Traffic accident analysis using machine learning paradigms". In: *Informatica* 29.1 (2005).

[19] Miao M Chong, Ajith Abraham, and Marcin Paprzycki. "Traffic accident analysis using decision trees and neural networks". In: *arXiv preprint cs/0405050* (2004).

[20] Sumit Chopra, Raia Hadsell, Yann LeCun, et al. "Learning a similarity metric discriminatively, with application to face verification". In: *CVPR (1)*. 2005, pp. 539–546.

[21] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification". In: *arXiv preprint arXiv:1202.2745* (2012).

[22] Marc Claesen et al. "Hyperparameter tuning in Python using Optunity". In: *Proceedings of the International Workshop on Technical Computing for Machine Learning and Mathematical Engineering*. Vol. 1. 2014, p. 3.

[23] Anton ML Coenen. "Neuronal activities underlying the electroencephalogram and evoked potentials of sleeping and waking: implications for information processing". In: *Neuroscience & Biobehavioral Reviews* 19.3 (1995), pp. 447–463.

[24] Jacques De Villiers and Etienne Barnard. "Backpropagation neural nets with one and two hidden layers". In: *IEEE transactions on neural networks* 4.1 (1993), pp. 136–141.

[25] Anthony J Devaney. "A filtered backpropagation algorithm for diffraction tomography". In: *Ultrasonic imaging* 4.4 (1982), pp. 336–350.

[26] R Glen Donaldson and Mark Kamstra. "An artificial neural network-GARCH model for international stock return volatility". In: *Journal of Empirical Finance* 4.1 (1997), pp. 17–46.

[27] Niklas Donges. "The random forest algorithm". In: *Towards Data Science. https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd* (2018).

[28] Rob A Dunne and Norm A Campbell. "On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function". In: *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*. Vol. 181. Citeseer. 1997, p. 185.

[29] Glyn Elwyn et al. “Decision analysis in patient care”. In: *The Lancet* 358.9281 (2001), pp. 571–574.

[30] Athanasios Episcopos and Jefferson Davis. “Predicting returns on Canadian exchange rates with artificial neural networks and EGARCH-M models”. In: *Neural Computing & Applications* 4.3 (1996), pp. 168–174.

[31] Brian Everitt and Anders Skrondal. “Standardized mortality rate (SMR)”. In: *The Cambridge Dictionary of Statistics* 409 (2010).

[32] Scott Fortmann-Roe. “Understanding the bias-variance tradeoff”. In: (2012).

[33] Yoav Freund and Llew Mason. “The alternating decision tree learning algorithm”. In: *icml*. Vol. 99. 1999, pp. 124–133.

[34] Mark A Friedl and Carla E Brodley. “Decision tree classification of land cover from remotely sensed data”. In: *Remote sensing of environment* 61.3 (1997), pp. 399–409.

[35] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. “Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)”. In: *The annals of statistics* 28.2 (2000), pp. 337–407.

[36] Jerome H Friedman. “Data Mining and Statistics: What’s the connection?” In: *Computing Science and Statistics* 29.1 (1998), pp. 3–9.

[37] Jerome H Friedman and Werner Stuetzle. “Projection pursuit regression”. In: *Journal of the American statistical Association* 76.376 (1981), pp. 817–823.

[38] Borivoje Furht and Armando Escalante. *Handbook of cloud computing*. Vol. 3. Springer, 2010.

[39] Tadayoshi Fushiki. “Estimation of prediction error by using K-fold cross-validation”. In: *Statistics and Computing* 21.2 (2011), pp. 137–146.

[40] YQ Gao et al. “Deep Residual Network with Transfer Learning for Imagebased Structural Damage Recognition”. In: *Eleventh US National Conference on Earthquake Engineering, Integrating Science, Engineering & Policy*. 2018.

[41] Yuqing Gao and Khalid M Mosalam. “Deep transfer learning for image-based structural damage recognition”. In: *Computer-Aided Civil and Infrastructure Engineering* 33.9 (2018), pp. 748–768.

[42] Stuart Geman, Elie Bienenstock, and René Doursat. “Neural networks and the bias/variance dilemma”. In: *Neural computation* 4.1 (1992), pp. 1–58.

[43] Marcel van Gerven and Sander Bohte. *Artificial neural networks as models of neural information processing*. Frontiers Media SA, 2018.

[44] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[45] Paulo B Goes. "Design science research in top information systems journals". In: *MIS Quarterly: Management Information Systems* 38.1 (2014), pp. iii–viii.

[46] JA Greenwood and JB Pl Williamson. "Contact of nominally flat surfaces". In: *Proceedings of the royal society of London. Series A. Mathematical and physical sciences* 295.1442 (1966), pp. 300–319.

[47] YB Guo and David W Yen. "A FEM study on mechanisms of discontinuous chip formation in hard machining". In: *Journal of Materials Processing Technology* 155 (2004), pp. 1350–1356.

[48] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection". In: *Journal of machine learning research* 3.Mar (2003), pp. 1157–1182.

[49] Martin T Hagan and Mohammad B Menhaj. "Training feedforward networks with the Marquardt algorithm". In: *IEEE transactions on Neural Networks* 5.6 (1994), pp. 989–993.

[50] Robert Haining. *Spatial data analysis in the social and environmental sciences*. Cambridge University Press, 1993.

[51] Alireza Hajian and Peter Styles. "Artificial neural networks". In: *Application of Soft Computing and Intelligent Methods in Geophysics*. Springer, 2018, pp. 3–69.

[52] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Unsupervised learning". In: *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[53] E Haug, T Scharnhorst, and P Du Bois. "FEM-Crash, Berechnung eines Fahrzeugfrontalaufpralls". In: *VDI Berichte* 613 (1986), pp. 479–505.

[54] Richard F Hespos and Paul A Strassmann. "Stochastic decision trees for the analysis of investment decisions". In: *Management Science* 11.10 (1965), B–244.

[55] Geoffrey E Hinton, Terrence Joseph Sejnowski, and Tomaso A Poggio. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[56] TK Ho. "Random decision forests (PDF): Proceedings of the 3rd International Conference on Document Analysis and Recognition". In: (1995).

[57] Daniel Hsu, Sham M Kakade, and Tong Zhang. "A spectral algorithm for learning hidden Markov models". In: *Journal of Computer and System Sciences* 78.5 (2012), pp. 1460–1480.

[58] Te-Ming Huang, Vojislav Kecman, and Ivica Kopriva. *Kernel based algorithms for mining huge data sets*. Vol. 1. Springer, 2006.

[59] ASCE Task Committee on Application of Artificial Neural Networks in Hydrology. "Artificial neural networks in hydrology. I: Preliminary concepts". In: *Journal of Hydrologic Engineering* 5.2 (2000), pp. 115–123.

[60] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[61] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[62] Gareth M James. "Variance and bias for general loss functions". In: *Machine Learning* 51.2 (2003), pp. 115–135.

[63] Gordon R Johnson and William H Cook. "Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures". In: *Engineering fracture mechanics* 21.1 (1985), pp. 31–48.

[64] A Jones et al. "Machine learning techniques to repurpose Uranium Ore Concentrate (UOC) industrial records and their application to nuclear forensic investigation". In: *Applied Geochemistry* 91 (2018), pp. 221–227.

[65] Iebeling Kaastra and Milton Boyd. "Designing a neural network for forecasting financial and economic time series". In: *Neurocomputing* 10.3 (1996), pp. 215–236.

[66] Barry L Kalman and Stan C Kwasny. "Why tanh: choosing a sigmoidal function". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 4. IEEE. 1992, pp. 578–581.

[67] Bogumił Kamiński, Michał Jakubczyk, and Przemysław Szufel. "A framework for sensitivity analysis of decision trees". In: *Central European journal of operations research* 26.1 (2018), pp. 135–159.

[68] WJ Kang et al. *Identification of dynamic behavior of sheet metals for an auto-body with tension split Hopkinson bar*. Tech. rep. SAE Technical Paper, 1998.

[69] Kamran Karimi and Howard J Hamilton. "Generation and interpretation of temporal decision rules". In: *arXiv preprint arXiv:1004.3334* (2010).

[70] Jungwon Kim et al. "Immune system approaches to intrusion detection–a review". In: *Natural computing* 6.4 (2007), pp. 413–466.

[71] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[72] Trenton Kirchdoerfer and Michael Ortiz. "Data-driven computational mechanics". In: *Computer Methods in Applied Mechanics and Engineering* 304 (2016), pp. 81–101.

[73] Trenton Kirchdoerfer and Michael Ortiz. "Data-driven computing in dynamics". In: *International Journal for Numerical Methods in Engineering* 113.11 (2018), pp. 1697–1710.

[74] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145.

[75] Andreas König et al. *Knowledge-Based and Intelligent Information and Engineering Systems, Part II: 15th International Conference, KES 2011, Kaiserslautern, Germany, September 12-14, 2011, Proceedings*. Vol. 6882. Springer, 2011.

[76] John R Koza et al. "Automated design of both the topology and sizing of analog electrical circuits using genetic programming". In: *Artificial Intelligence in Design96*. Springer, 1996, pp. 151–170.

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[78] Ray Kurzweil. *The singularity is near: When humans transcend biology*. Penguin, 2005.

[79] Doug Laney. "3D data management: Controlling data volume, velocity and variety". In: *META group research note* 6.70 (2001), p. 1.

[80] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[81] Tae-Hwy Lee, Halbert White, and Clive WJ Granger. "Testing for neglected nonlinearity in time series models: A comparison of neural network methods and alternative tests". In: *Journal of Econometrics* 56.3 (1993), pp. 269–290.

[82] Xin Lei et al. "Machine Learning-Driven Real-Time Topology Optimization Under Moving Morphable Component-Based Framework". In: *Journal of Applied Mechanics* 86.1 (2019), p. 011004.

[83] David J Leinweber. "Stupid data miner tricks: overfitting the S&P 500". In: *Journal of Investing* 16.1 (2007), p. 15.

[84] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[85] Thanasis Loupas, WN McDicken, and Paul L Allan. "An adaptive weighted median filter for speckle suppression in medical ultrasonic images". In: *IEEE transactions on Circuits and Systems* 36.1 (1989), pp. 129–135.

[86] Computing Machinery. "Computing machinery and intelligence-AM Turing". In: *Mind* 59.236 (1950), p. 433.

[87] Jahnavi Mahanta. "Introduction to Neural Networks, Advantages and Applications". In: *Towards Data Science* (2017).

[88] LS-DYNA Keyword Users Manual. "Version 960". In: *Livermore Software Technology Corporation* (2001).

[89] André C Marreiros et al. "Population dynamics: variance and the sigmoid activation function". In: *Neuroimage* 42.1 (2008), pp. 147–157.

[90] José M Matías et al. "Boosting GARCH and neural networks for the prediction of heteroskedastic time series". In: *Mathematical and Computer Modelling* 51.3-4 (2010), pp. 256–271.

[91] Jesus Mena. *Machine learning forensics for law enforcement, security, and intelligence*. Auerbach Publications, 2016.

[92] Nasser M Nasrabadi. "Pattern recognition and machine learning". In: *Journal of electronic imaging* 16.4 (2007), p. 049901.

[93] N Nayab and J Scheid. *Disadvantages to using decision trees*. 2015.

[94] Nashreen Nesa, Tania Ghosh, and Indrajit Banerjee. "Non-parametric sequence-based learning approach for outlier detection in IoT". In: *Future Generation Computer Systems* 82 (2018), pp. 412–421.

[95] Andrzej S Nowak and Kevin R Collins. *Reliability of structures*. CRC Press, 2012.

[96] Noel M O'Boyle et al. "Open Babel: An open chemical toolbox". In: *Journal of cheminformatics* 3.1 (2011), p. 33.

[97] Chee-Mun Ong et al. *Dynamic simulation of electric machinery: using MATLAB/SIMULINK*. Vol. 5. Prentice hall PTR Upper Saddle River, NJ, 1998.

[98] JH Park et al. "Economic load dispatch for piecewise quadratic cost function using Hopfield neural network". In: *IEEE transactions on power systems* 8.3 (1993), pp. 1030–1038.

[99] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[100] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions". In: (2018).

[101] Paola Rizzoli et al. "Relative height error analysis of TanDEM-X elevation data". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 73 (2012), pp. 30–38.

[102] Stefan Rolewicz. *Functional analysis and control theory: linear systems*. Vol. 29. Springer Science & Business Media, 2013.

[103] Matthias Rupp. "Machine learning for quantum mechanics in a nutshell". In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1058–1073.

[104] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[105] S Rasoul Safavian and David Landgrebe. "A survey of decision tree classifier methodology". In: *IEEE transactions on systems, man, and cybernetics* 21.3 (1991), pp. 660–674.

[106] Paul Sajda. "Machine learning for detection and diagnosis of disease". In: *Annu. Rev. Biomed. Eng.* 8 (2006), pp. 537–565.

[107] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[108] Adam Santoro et al. "Meta-learning with memory-augmented neural networks". In: *International conference on machine learning*. 2016, pp. 1842–1850.

[109] Fabrizio Sebastiani. "Machine learning in automated text categorization". In: *ACM computing surveys (CSUR)* 34.1 (2002), pp. 1–47.

[110] Giovanni Seni and John F Elder. "Ensemble methods in data mining: improving accuracy through combining predictions". In: *Synthesis lectures on data mining and knowledge discovery* 2.1 (2010), pp. 1–126.

[111] Noam Shazeer et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer". In: *arXiv preprint arXiv:1701.06538* (2017).

[112] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[113] DASSAULT Simulia. "ABAQUS 6.11 analysis user's manual". In: *Abaqus* 6 (2011), pp. 22–2.

[114] Shailesh K Singh et al. "Experimental and numerical studies on friction welding of thixocast A356 aluminum alloy". In: *Acta Materialia* 73 (2014), pp. 177–185.

[115] Michael R Smith and Tony Martinez. "Improving classification accuracy by identifying and removing instances that should be misclassified". In: *The 2011 International Joint Conference on Neural Networks*. IEEE. 2011, pp. 2690–2697.

[116] Donald F Specht. "A general regression neural network". In: *IEEE transactions on neural networks* 2.6 (1991), pp. 568–576.

[117] Donald F Specht. "Probabilistic neural networks". In: *Neural networks* 3.1 (1990), pp. 109–118.

[118] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[119] Daniel P Steinfort et al. "Cost-benefit of minimally invasive staging of non-small cell lung cancer: a decision tree sensitivity analysis". In: *Journal of Thoracic Oncology* 5.10 (2010), pp. 1564–1570.

[120] Carolin Strobl, James Malley, and Gerhard Tutz. "An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests." In: *Psychological methods* 14.4 (2009), p. 323.

[121] Toshiyasu Tarumi et al. "Infinite impulse response filters for direct analysis of interferogram data from airborne passive Fourier transform infrared spectrometry". In: *Vibrational spectroscopy* 37.1 (2005), pp. 39–52.

[122] Sebastian Thrun, Lawrence K Saul, and Bernhard Schölkopf. *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*. Vol. 16. MIT press, 2004.

[123] Julius T Tou and Rafael C Gonzalez. "Pattern recognition principles". In: (1974).

[124] Jack V Tu. "Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes". In: *Journal of clinical epidemiology* 49.11 (1996), pp. 1225–1231.

[125] Jason Weston, Chris Watkins, et al. "Support vector machines for multi-class pattern recognition." In: *Esann*. Vol. 99. 1999, pp. 219–224.

[126] Wiphada Wettayaprasit, Nasith Laosen, and Salinla Chevakidagarn. "Data filtering technique for neural networks forecasting". In: *Proceedings of the 7th WSEAS International Conference on Simulation, Modelling and Optimization*. World Scientific, Engineering Academy, and Society (WSEAS). 2007, pp. 225–230.

[127] Wiphada Wettayaprasit and Pornpimon Nanakorn. "Feature extraction and interval filtering technique for time-series forecasting using neural networks". In: *2006 IEEE Conference on Cybernetics and Intelligent Systems*. IEEE. 2006, pp. 1–6.

[128] John C Wooley, Herbert S Lin, National Research Council, et al. "Computational modeling and simulation as enablers for biological discovery". In: *Catalyzing inquiry at the interface of computing and biology*. National Academies Press (US), 2005.

[129] Fen Wu et al. "Induced L2-norm control for LPV systems with bounded parameter variation rates". In: *International Journal of Robust and Nonlinear Control* 6.9-10 (1996), pp. 983–998.

[130] Ping Zhang. "Model selection via multifold cross validation". In: *The Annals of Statistics* (1993), pp. 299–313.

[131] Tong Zhang. "An introduction to support vector machines and other kernel-based learning methods". In: *AI Magazine* 22.2 (2001), p. 103.

[132] Xiaolong Zheng, Peng Zheng, and Rui-Zhi Zhang. "Machine learning material properties from the periodic table using convolutional neural networks". In: *Chemical Science* 9.44 (2018), pp. 8426–8432.

[133] Quan Zhou et al. "Learning atoms for materials discovery". In: *Proceedings of the National Academy of Sciences* 115.28 (2018), E6411–E6417.