THE UNIVERSITY *of* EDINBURGH

Using Machine Learning to predict the ballistic response of structures to projectile impact

THE UNIVERSITY
of EDINBURGH

Samuel Thompson

31st October 2021

Samuel Thompson MEng

Using machine learning to predict the ballistic response of structures to projectile impact

August 31st, 2021

Reviewers: Dr Shannon Ryan and Dr Encarni Medina-Lopez

Supervisors: Dr Filipe Teixeira-Dias and Dr Mariana Paulino

The University of Edinburgh

Institute for Infrastructure and Environment

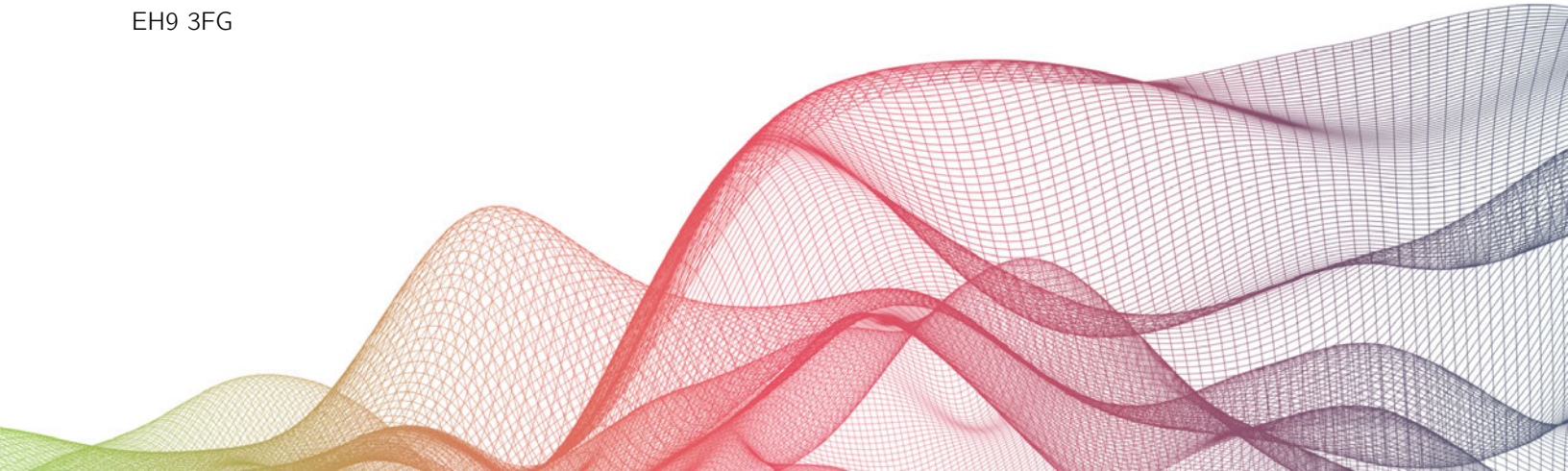
School of Engineering, The University of Edinburgh

William Rankine Building, The King's Buildings

Thomas Bayes Road

Edinburgh

EH9 3FG



Abstract

Ballistic loading is a primary risk in both civil and military defence applications, where successfully predicting the dynamic response of a material to impact is a fundamental component of the design of safe and fit-for-purpose protective structures. Approaches to understand the response to ballistic impact conventionally revolve around experimental tests, whereby the material or structure of interest is subject to impact by a projectile across a range of impact velocities. However, experimental testing is expensive and incurs large costs due to the destructive nature of the testing and the specialist equipment required. Numerical tools, such as the Finite Element (FE) method, play an important role by filling the gaps left sparse by experimental results and contribute towards the complete dynamic material characterisation campaign. This thesis considers an alternative to FE models by using Machine Learning (ML) techniques that learn directly from the available ballistic data. Specifically, the thesis considers the use of Multi-Layer Perceptron (MLP) models to predict the ballistic response of multi-layered targets to impact but its primary intention is to explore the value that generative networks can bring to the ballistic domain. This thesis shows how Generative Adversarial Networks (GANs) can be used to supplement sparse ballistic datasets by generating new samples representative of the dataset that it was trained on, but also how they can be used to predict key ballistic parameters for engineering design such as the ballistic limit velocity, v_{bl}. And finally, how conditional-GANs (cGANS) can be utilised to allow the network to be conditioned on additional auxiliary information such as class labels that refer to a specific property relevant to the ballistic data thus allowing the cGAN to generate new samples specific to the class label given. This allows the trained cGAN to generate data for classes that are not present in the training set and conduct its own material characteristic campaign. The justification for using ML practices for in the ballistic domain lies in the idea that numerical models are adjusted such that the output is consistent with the results from experimental testing. There is therefore an opportunity for research to explore whether ML techniques can capture that same distribution by training on the ballistic data directly.

Acknowledgements

First of all, this thesis simply would not exist without my supervisor Dr. Filipe Teixeira Dias. I have known Filipe for almost 7 years now and had the pleasure of also completing my Master's project with him. He is the only person that I have ever met that shares the belief that an extra 3 hours spent formatting a graph that already looked fine is a worthwhile endeavour. More than anything I appreciate the confidence he has in me and the freedom he gave throughout the duration of the PhD. I believe that few supervisors would let their student switch projects entirely and start fresh 16 months in, let alone to a topic that they have no experience in themselves, but I'm grateful that I found one that would. I have learned that a PhD is a very unique experience, I have friends that have had a vastly different experience to mine, and I have to take this opportunity to thank Filipe directly for making the entire process so enjoyable and stress-free. A huge thank you also has to go to my second supervisor Dr. Mariana Paulino. Despite being on the opposite side of the world she still managed to find the time to join meetings regularly, albeit to remind us how she is able to have BBQ's and sunbathe whilst it is snowing for us. I would also like to extend thanks to Maisie and Dr. Adam Robinson along with a massive thank you to Dr Shannon Ryan and Dr Encarni Medina-Lopez - their considerate comments and suggestions truly helped improve the quality of my thesis for which I am extremely grateful. Finally, the Engineering and Physical Sciences Research Council (EPSRC) funded this work through Grant EP/N509644/1. Without their support, this PhD and many others simply would not be possible.

On the non-academic side of things, a huge thank you in particular to Alastair, Charlotte, Jazim, and Aldo for being along for the ride during the PhD. I'd also like to dedicate this thesis to Harry, who has had a pretty wild couple of years to say the least, but still manages to inspire me like nothing else. Thanks to the *Real Bro's of Tameside* for keeping me entertained the entire time and okay, fine - thanks to Evan as well. In addition, I need to thank everybody at RYBBA. It simply would not be possible to thank all of my training partners directly, but who knew that *murder yoga* would be the perfect distraction to rest my mind outside of research. When most evenings are spent figuring out the most effective ways to break peoples limb's and how to stop them doing the same to you, suddenly the thought of completing a PhD is much less intimidating.

Finally it goes without saying that I want to thank my Mum, Dad, Lauren, Emily, and especially Theresa, for being the best throughout the whole process.

Samuel Thompson
Edinburgh, Oct 31st, 2021

Declaration

I, Samuel Thompson, declare that this thesis titled, '*Using Machine Learning to predict the ballistic response of structures to projectile impact*' and the work presented in it are my own. I also declare that this thesis has been partially published in a book chapter and several journal papers, where appropriate, this has been stated at the beginning of each relevant chapter in the thesis where published content has been used I confirm that:

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

Samuel Thompson
Edinburgh, Aug 31st, 2021

Justification

I should state that the origin of this research came around the idea of trying to address a problem that exists within this space – and that problem is that ballistic data is often sparse, of low quality and expensive to produce. This makes the prediction of the ballistic limit velocity (BLV) an active area of research in the field of materials science and engineering. There are a variety of methods and models that have been developed to predict the BLV of different materials and configurations, each with its own advantages and limitations. I will begin by giving a quick overview of current approaches to this problem, before offering a justification as to why I believe that this research offers some benefit.

One approach is to use analytical models to describe the behaviour of the target plate under high strain rates and subsequently estimate the BLV based on the material properties and geometry of the plate. In practice however, there is often a discrepancy between predictions made by the analytical model and the actual ballistic response of a target plate due to external factors that are not accounted for in the analytical models. Another common approach is to use numerical simulations, such as finite element analysis (FEA) to model the behaviour of the target plate under ballistic impact. These simulations are based on first principles and can provide a detailed understanding of the deformation and failure mechanisms of the plate. They are often used to optimise the design of the plate for specific applications. However, FEA requires accurate material models, complex meshing and the quality of the output is often dependent on the assumptions made when creating the model. They are also computationally expensive and require a great deal of time and effort by a specialised professional to assure that the predictions from the model are realistic. In addition, such models are developed in combination with experimental testing and are fine-tuned on an ad-hoc basis such that its output is in alignment with the experimental results. Once a model has been developed, if we then wish to test the ballistic response of a plate at a different impact velocity or plate thickness, the model must be ran again and there is therefore a computational component that governs how quickly predictions can be retrieved.

Finally, experimental testing is the most reliable method at our disposal to determine the ballistic response of a material, albeit the slowest and most expensive. Experimental techniques have progressed to include high-speed imaging and digital image correlation which have enabled a more detailed analysis of the behaviour of materials under high strain rates. Such experimental testing however is expensive, and the destructive nature of ballistic testing only increases the cost. Specialised equipment, personnel and a finite amount of target samples limit the amount of testing that can be performed and subsequently the completeness of material characterisation campaigns.

The primary hypothesis of this thesis is to consider whether ML methods could provide an

alternative way of predicting the BLV. ML has been an active topic of research in the field of material science and there are multiple accounts of ML models being applied specifically to this problem space. Whilst reviewing the literature however, to my knowledge I could not find any evidence of researchers considering generative models to supplement sparse datasets. The question that I wished to answer was:

> "If experimental results are used as the foundation against which numerical models are fine-tuned, is there an opportunity to test whether generative ML models are able to learn the distribution of a ballistic dataset and supplement that dataset with additional ballistic samples".

I believed that this would be a useful endeavour, especially given the exhaustive process that is currently required to achieve ballistic limit velocity predictions. If it is in fact possible to utilise a generative model to produce synthetic data that is accurate and representative of actual ballistic data – then many additional samples and predictions can be obtained instantly. Using GANs to supplement sparse datasets is not isolated in the literature. In 2018, NVIDIA proposed a method to generate synthetic abnormal MRI images with brain tumours by training a GAN using two datasets available to the public of brain MRI Imagery. They demonstrated two key benefits from the synthetic data; the first is that they observed improved performance on tumour segmentation by leveraging the synthetic images as a form of data augmentation. They utilised a GAN to supplement the available data to yield additional benefits from their other ML models. The second is that they demonstrated comparable tumour segmentation results when trained on the synthetic data versus when trained on the real subject data. They declared that such results offer a potential solution to two of the largest challenges facing machine learning in medical imaging such as the small incidence of pathological findings and the restrictions around sharing patient data. I believe that this problem is analogous to that of ballistic data given the sparsity of it and the low quality. I also believe that there is an opportunity to augment ballistic datasets using generative models to facilitate the use of other ML models to make additional predictions.

I would also like to state clearly that I do not believe that I have solved the problem of generating synthetic ballistic data, but I do believe that this thesis acts as a first step in that direction and I would like to believe that it might act as a foundation on which to explore other ML methods within this space. This methodology, however, must be tested more thoroughly and applied on a wider range of problems across a richer variety of ballistic datasets. Of course, there is also an argument that the predictions made by ML models are a black box and not built upon first principles – however I believe that if it can be demonstrated that ML models can consistently produce accurate results across a range of problems and configurations, then the scientific community might increase their opinion on the validity of the results obtained in this fashion. To extend upon this point, black box concerns might be eased somewhat by the

implementation of physics-informed machine learning models. Here the ML model could be initialised with analytical models that establish the rules of the game, in other words, define the boundary of the problem such that the model has an understanding of those first principles. Transfer learning could then be utilised to re-train such a model on a specific experimental dataset in an attempt to 1) reduce the amount of data required to train such models and 2) help generalise the output of the model such that it can generate synthetic data across a wider range of impact velocities which may not be present in the training set.

Contents

1 Introduction **1**

1.1 Motivation . . . 1
1.2 Objectives . . . 2
1.3 Thesis Structure . . . 3
1.4 An overview of experimental ballistic testing . . . 5
1.5 Analytical Modelling . . . 8
1.6 Numerical Modelling . . . 9
1.7 Machine learning . . . 12

2 Machine Learning **16**

2.1 Preface . . . 16
2.2 Introduction . . . 17
2.2.1 The Perceptron . . . 19
2.3 Activation Functions . . . 22
2.3.1 Overview . . . 22
2.3.2 Sigmoid and tanh Activation Functions . . . 23
2.3.3 Rectified Linear Unit Activation Functions . . . 25
2.3.4 Softmax Activation Function . . . 27
2.4 Optimisers and Loss Functions . . . 29
2.4.1 Optimiser algorithms . . . 30
2.4.2 Regression and Classification loss functions . . . 36
2.5 Evaluation of Optimisers and Loss Functions . . . 38
2.5.1 Performance evaluation of optimisers . . . 38
2.5.2 Performance evaluation of loss functions for classification . . . 40
2.6 LSTM Networks . . . 43
2.6.1 Overview . . . 43
2.6.2 LSTM Layer Architecture . . . 44
2.7 Predicting the response of a SDOF system . . . 48
2.7.1 Establishing the training set . . . 49
2.7.2 Setup of the LSTM Model . . . 50

2.7.3 Results . . . 52
2.7.4 Conclusion . . . 54
2.8 Generative Adversarial Networks . . . 56
2.8.1 Training a GAN . . . 58
2.8.2 Example: Training a GAN network on a sine wave . . . 60

3 Perforation of Multi Layered Targets 63

3.1 Introduction . . . 64
3.1.1 The hybrid methodology . . . 65
3.2 Analytical modelling . . . 65
3.3 ML model . . . 69
3.3.1 Problem setting . . . 69
3.3.2 AI setup . . . 70
3.4 Output from the MLP model . . . 72
3.5 Finite element modelling . . . 77
3.5.1 Model information . . . 77
3.5.2 Single layer test cases . . . 78
3.5.3 Multi-layered targets . . . 79
3.6 Concluding remarks . . . 81

4 Ballistic response of armour plates using GANs 84

4.1 Generative Adversarial Networks . . . 85
4.1.1 Training sets . . . 86
4.1.2 Model architecture . . . 88
4.1.3 Training algorithm . . . 89
4.1.4 Model evaluation . . . 91
4.2 Results from GAN model . . . 92
4.3 Concluding remarks . . . 97

5 Predictions on multi-class ballistic datasets using cGANs 100

5.1 Methodology . . . 101
5.1.1 Conditional Generative Adversarial Networks . . . 102
5.1.2 Model evaluation . . . 108
5.2 Results and Discussion . . . 111
5.2.1 Non-linear least squares ballistic limit velocity predictions . . . 113
5.2.2 Model variation analysis and distributions . . . 116
5.2.3 Evaluation of cGAN models on unseen class labels . . . 117
5.3 Concluding remarks . . . 123

6 Supplementing experimental datasets using GANs **124**

6.1 Methodology . . . 125
6.1.1 Training Set . . . 125
6.1.2 Model Architecture . . . 126
6.1.3 Training algorithm . . . 127
6.2 Results . . . 127
6.2.1 Training evaluation and discussion . . . 128
6.3 Conclusion . . . 129

7 Future work and physics informed machine learning **133**

7.1 Variational Autoencoders . . . 134
7.1.1 VAE Architecture . . . 134
7.1.2 Discussion . . . 136

8 Thesis Concluding Remarks **140**

8.1 Significance and Implications . . . 140
8.2 Contribution to knowledge . . . 142
8.3 Future Work . . . 143

Nomenclature

Acronyms and abbreviations

AdaGrad	Adaptive Gradient
AI	Artificial Intelligence
AM	Additive Manufacturing
ANN	Artificial Neural Network
AP	Armour Piercing
BCE	Binary Cross Entropy
BLV	Ballistic Limit Velocity
cGAN	Conditional Generative Adversarial Network
CIFAR	Canadian Institute for Advanced Research
CL	Cockroft-Latham
CNN	Convolutional Neural Network
DL	Deep Learning
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
GAN	Generative Adversarial Network
IDC	International Data Corporation
JC	Johnson-Cook
JH	Johnson-Holmquist
LM	Levenberg Marquadt
LReLU	Leaky Rectified Linear Unit
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MJC	Modified Johnson-Cook
ML	Machine Learning
MLP	Multi Layer Perceptron
MNIST	Modified National Institute of Standards and Technology
MPSE	Materials, Processes and Structural Engineering
MSE	Mean Squared Error
NLP	Natural Language Programming
PReLU	Parametric Rectified Linear Unit
ReLU	Rectified Linear Unit
RI	Recht-Ipson
RLReLU	Randomised Leaky Rectified Linear Unit
RMSE	Root Mean Square Error

RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
SDOF	Single Degree Of Freedom
SGD	Stochastic Gradient Descent
SHM	Structural Health Monitoring
SPH	Smooth Particle Hydrodynamics
SRGAN	Super Resolution Generative Adversarial Network
SVM	Support Vector Machines
TS	Training Set
UHWMPE	Ultra High Molecular Weight Polyethylene

Variables and symbols

a	Lambert parameter that governs shape of ballistic curve
a^i	Deceleration of projectile perforating layer i
$a_{\%}$	Percent difference between models prediction and Lambert parameter a
A	Yield stress
b	Trainable parameter bias
c	Damping coefficient
c_p	Elastic wave velocity of the projectile
c_s	Specific heat capacity
c_t	Elastic wave velocity of the target
$\mathbf{c}_\mathrm{t}$	Cell state at time step t
$\mathbf{C}$	LSTM Features
C	Strain rate sensitivity term derived from experimental tests
C_i	Empirically determined plastic strain parameter
d	Minimum distance between generated sample and ballistic curve
d_p	Projectile diameter
$\bar{d}$	Average minimum distance for generated samples
D	Discriminator model
$D(\mathbf{X})$	Discriminator model prediction on real data
$D(G(\mathbf{Z}))$	Discriminator model prediction on samples from the generator
D_c	Damage variable
E_X	Expected value over all real data instances
E_fn	Energy lost due to deformation and heat during impact
E_kg	Kinetic energy of the plug after impact
E_kp	Kinetic energy of the projectile after impact
E_in	Initial energy of a given system
E_out	Final energy of a given system

E_p	Young's modulus of projectile
E_t	Young's modulus of target
E_Z	Expected value over all fake data instances
f	Excitation of system
f_t	Forget gate
F	Activation function
F_g	Gate activation function
g_t	Cell candidate
$\mathbf{g}$	Gradient estimate
G	Generator model
$G(\mathbf{Z})$	Output of generator given latent input
h	Plate thickness
$\mathbf{h}_\mathrm{t}$	Hidden state at time step t
i	Current iteration
i_t	Input gate
k	Stiffness
$\mathbf{L}_\mathrm{x}$	Logits
L	Loss function
L_CE	Cross-entropy loss function
L_H	Hinge loss function
L_H2	Hinge square loss function
m	Size of minibatch
M	Thermal softening term derived from experimental tests
M_g	Mass of the plug (material from the target)
M_p	Mass of the projectile
n	Number of samples
N_i	Number of hidden units
o_t	Output gate
p	Lambert parameter that governs shape of ballistic curve
$p_\%$	Percent difference between models prediction and Lambert parameter p
$\mathbf{P}$	Probabilities
P_Z	Gaussian latent noise distribution
Q	Approximation of target probability by model
Q_i	Empirically determined plastic strain parameter
$\mathbf{r}$	Gradient accumulation variable for second-order moments
$\hat{\mathbf{r}}$	Corrected bias in second-order moments
$\mathbf{R}$	Recurrent weights
R	Isotropic strain hardening term
$\mathbf{s}$	Gradient accumulation variable for first-order moments
$\hat{\mathbf{s}}$	Corrected bias in first-order moments

S	Softmax function
t	Time step
t^{i}	Projectile contact time with layer i
T	Number of epochs
T_{r}	Room temperature
T_{m}	Melting temperature
$\dot{T}$	Temperature increment due to adiabatic heating
$T^{\star}$	Room temperature
u	Displacement
$\dot{u}$	Velocity
$\ddot{u}$	Acceleration
u_0	Initial displacement
$\dot{u}_0$	Initial velocity
v_{bl}	Ballistic limit velocity
$v_{\mathrm{bl\%}}$	Percent difference between models prediction and Lambert parameter v_{bl}
v_{f}	Free impact final velocity
v_{i}	Impact velocity of a projectile fired at a target
v_{r}	Residual velocity of a projectile after perforating a target
V	Relative Velocity
$\mathbf{w}$	Trainable weights in ANN
$\mathbf{W}$	Input Weights
W_{c}	Failure parameter
W_{n}	Loss of work (energy) due to adiabatic shearing
$\mathbf{X}$	Training data
$\bar{y}$	Mean average of data output by trained model
y_{max}	Maximum value of data output by trained model
y_{min}	Minimum value of data output by trained model
$\hat{y}$	Prediction from model
$\mathbf{Y}$	Target data
$\mathbf{Z}$	Latent input to Generator model

Greek symbols

α	Parameter to determine negative gradient of LReLU function
β	Parameter to define variation of acceleration over particular time step
β_1	Exponential decay rate for first-order moment estimate
β_2	Exponential decay rate for second-order moment estimate
γ	Momentum term

δ	Term included in optimisation algorithms to prevent division by 0
ϵ	Learning rate
ϵ_{p}	Equivalent strain
$\boldsymbol{\theta}$	Trainable parameters of ANN
λ	Class label
μ_{k}	Coefficients of kinetic friction
ν_{t}	Poisson's ratio
ξ	Momentum hyperparameter
ρ_{p}	Density of projectile
ρ_{t}	Density of target
σ	Standard deviation
σ_{p}	Equivalent stress
τ	Parameter to control length scale of moving average in RMSProp
Υ	Secondary parameter to define variation of acceleration over particular time step
χ	Taylor-Quinney coefficient
φ	Yield Constant

Chapter 1

Introduction

1.1 Motivation

Machine learning (ML) models, which have demonstrated tremendous success in commercial applications such as home voice assistants and self-driving cars [Gupta et al., 2021, Sherstinsky, 2020], are beginning to play an important role in advancing scientific discovery in domains traditionally dominated by mechanistic models; such as those derived from first principles [Willard et al., 2020]. There is a growing consensus within the scientific community that suggests that neither a ML-only nor a scientific knowledge-only approach can be considered sufficient for complex scientific and engineering applications and it is becoming more common for researchers to explore the continuum between mechanistic and ML models where there is a symbiotic relationship between the integration of scientific knowledge and effective data practices [Alber et al., 2019, Baker et al., 2019, Karpatne et al., 2016, Rai and Sahu, 2020, Schleder et al., 2019]. Despite this idea growing momentum in recent years [Karpatne et al., 2016], a vast amount of research integrating scientific and ML principles has already been pursued across a diverse range of disciplines such as biological sciences [Lu et al., 2020], climate sciences [Faghmous and Kumar, 2014, O'Gorman and Dwyer, 2018] and turbulence modelling [Bode et al., 2019, Mohan and Gaitonde, 2018]. Yet materials and structures engineers have been slower to engage with these advancements. [Dimiduk et al., 2018] suggest that the recent advances that are driving other technical fields are not sufficiently distinguished from long-known informatics methods for materials and subsequently masking their likely impact to the materials, processes, and structures engineering (MPSE) fields. Additionally, the diverse nature and limited availability of relevant materials data pose obstacles to ML application.

This thesis considers the issue of predicting the response of materials to ballistic impacts.

Predicting the outcome of impulsive events such as terminal ballistic impact is a complex problem due to the difficulties in characterising material behaviour across a wide range of loading rates and impact scenarios. In terms of analysis, it is the combined sum and influence of a number of input variables that ultimately govern the response of a material and structure under a specific loading scenario [Ryan et al., 2016]. Ballistic loading is a primary risk in both civil and military defence applications, where successfully predicting the dynamic response of a material is a fundamental component of the design of safe and fit-for-purpose protective structures. Approaches to understand the response to ballistic impact typically revolve around experimental tests, whereby the material or structure of interest is subject to impact by a projectile in a controlled environment across a range of impact velocities [Børvik et al., 1999a, Børvik et al., 2001b, Børvik et al., 2003, Børvik et al., 2005, Huang et al., 2018, Rosenberg et al., 2016, Sikarwar et al., 2014, Wei et al., 2012]. However, the need for specialist equipment such as high-speed cameras combined with the destructive nature of experimental testing incurs large costs for each of the experimental processes involved. This cost is magnified when evaluating the response of complex composites or other materials that are also expensive to fabricate/source. Dynamic material characterisation campaigns often accompany ballistic tests in order to develop material models for numerical simulations of the penetration and perforation processes. Numerical tools, such as the Finite Element (FE) method, play an important role by filling the gaps left sparse by experimental results and the results of which are widely available in the literature [Feng et al., 2020, Scazzosi et al., 2021, Xu et al., 2019]. Numerical models can then be used to modify design parameters and subsequently predict the response of the material given new loading conditions. FE simulations often require large computational resources and the parameters of the model are modified such that an agreement with the experimental data is met [Gonzalez-Carrasco et al., 2011].

1.2 Objectives

Instead of using the traditional approach of complementing experimental testing with numerical studies, the primary goal of this thesis considers the feasibility of predicting key ballistic parameters such as the ballistic limit velocity, v_{bl}, by training generative networks on ballistic datasets directly. The intention is to utilise ML techniques to fill the gaps left sparse by experimental testing instead of relying on conventional numerical methods. The justification for this approach lies in the idea that numerical models are adjusted such that the output is consistent with the results from experimental testing. There is therefore an opportunity for research to explore whether ML techniques can capture that same distribution by training on the ballistic data directly. To that end, this thesis considers the following objectives:

1. To demonstrate how machine learning can be used to predict the ballistic response of

multi-layered targets.

2. To propose a novel use of generative networks to supplement sparse ballistic datasets and predict key ballistic parameters.

3. To propose a novel approach using conditional generative networks to conduct material characterisation campaigns.

1.3 Thesis Structure

This thesis has been organised into 7 chapters which contain the following:

Chapter 1 introduces the general problem setting for the thesis and the key objectives that the thesis aims to fulfill. It highlights some of the issues that are associated with ballistic testing with regards to the design of safe and fit-for-purpose structures and the prediction of key design parameters such as the ballistic limit velocity. The areas of targeted research that exist in the literature to address them are presented with regards to experimental, analytical and numerical modelling approaches. The chapter also includes the justification for introducing machine learning methods to tackle the same challenges and how they can be used to predict the ballistic response of structures and reflects upon the work that currently exists within the literature.

Chapter 2 provides an overview and explanation of some of the key machine learning principles that are used within the main body of work of the thesis. The section begins by conceptualising what machine learning is and where it places within the space of scientific research. The chapter introduces some basic concepts of machine learning from Multi Layer Perceptron (MLP) networks including the trainable parameters such as the weights and biases of a model and the key hyperparameters that must be defined. The chapter benchmarks the performance of Adam, RMSProp, AdaGrad and Stochastic Gradient Descent optimisers and cross entropy and hinge loss functions for classification tasks on the MNIST and CIFAR10 datasets. The use and implementation of activation functions is discussed before introducing recurrent networks, specifically long short-term memory (LSTM) networks, and their suitability for use with time series data. The LSTM model is then applied to estimate the dynamic response of a single degree of freedom (SDOF) system to extend upon the work of Wu et al. [Wu and Jahanshahi, 2019].Finally, the chapter details the workings and training procedure of Generative Adversarial Networks (GANs) which comprise the main body of work of this thesis. A simple example is presented that considers the model architecture and training procedure of how a GAN can be used to learn from and generate samples from the same distribution as a sin curve.

Chapter 3 proposes a hybrid method to study the perforation of multi-layered targets. The method combines an energy-based analytical approach with a set of deep learning models. Finite Element Analysis (FEA) and experimental results are used to train MLP models and validate the design process. The energy-based analytical method generates solutions for the ML algorithms with the intention of finding optimal configurations for the protective structure. The proposed MLP architecture is trained using both experimental results and analytical data to understand the ballistic response of a specific material and subsequently predict the residual velocity for a given impact velocity, layer thickness and material properties. Networks trained for individual layers of the armour system are then connected in order to predict the residual velocity of blunt projectiles perforating multi-layered composite structures. Validation tests are done on systems including both single and multi-layered targets.

Chapter 4 considers the ballistic response of armour plates using GANs. This study tests the feasibility of using GANs to make ballistic limit velocity predictions and considers three separate GAN networks each trained on a unique dataset created using the Lambert and Jonas ballistic model. In total, three training sets of degrading structural quality were used to train each of the models. 100, 50 and 10 samples were found in training sets 1, 2, and 3 respectively, where a single sample refers to an impact velocity and its corresponding residual velocity. Training set 3 was afflicted with an additional 10% noise to mimic that of measurement error. The Chapter shows how GANs can be used to supplement sparse ballistic datasets and the output from the trained GAN networks is compared and evaluated with respect to expected values.

Chapter 5 expands upon the work from Chapter 4 and documents the training and implementation of a conditional-GAN (cGAN) architecture. The cGAN addresses a common limitation of conventional GAN networks where there is limited governance over the output. cGANs can be conditioned by extra information during training such as class labels or data from other modalities to grant additional control over the output. In this study, the cGAN is conditioned by a class label that allows the algorithm to make a connection between certain distributions of data and that particular class label. The intention being that once the cGAN is trained, it can be used to generate new samples of data representative of the distribution mapped to that particular class label. To that end, the cGAN network is trained on a multi-class ballistic dataset containing 10 classes where each class corresponds to a ballistic curve with a different ballistic limit velocity. This dataset is analagous to that conducted during material characterisation campaigns during experimental testing that considers the ballistic response of a specific material at different thicknesses. This study demonstrates that cGAN networks can be used to allow local control over the output, allowing for additional class-specific samples to be generated with high accuracy and subsequently make predictions regarding the ballistic limit velocity for different classes. But more interestingly, the cGAN network can also be used to make ballistic predictions for ballistic classes that do not appear in the training set.

Chapter 6 applies the GAN model from Chapter 4 on real experimental data and evaluates its

ability to supplement ballistic datasets. A total of 4 GAN models are trained on 4 experimental datasets carried out by Costas et al. [Costas et al., 2021] which correspond to a steel bullet core and AP bullet impacting both an as-printed plate and a heat-treated plate.

Chapter 7 presents an alternative generative model known as the Variational Autoencoder (VAE) and demonstrates how it can be used to generate synthetic ballistic data after being trained on a training set produced by an anlytical model. The chapter then introduces physics-informed machine learning and presents a methodology to improve the generalisation of generative models for the synthesis of ballistic data by replacing the Generator of a GAN or cGAN model with a pre-trained decoder model.

Finally, Chapter 8 outlines the significance and implications of this work. Additionally, the limitations of the proposed methods are given along with recommendations for future work.

1.4 An overview of experimental ballistic testing

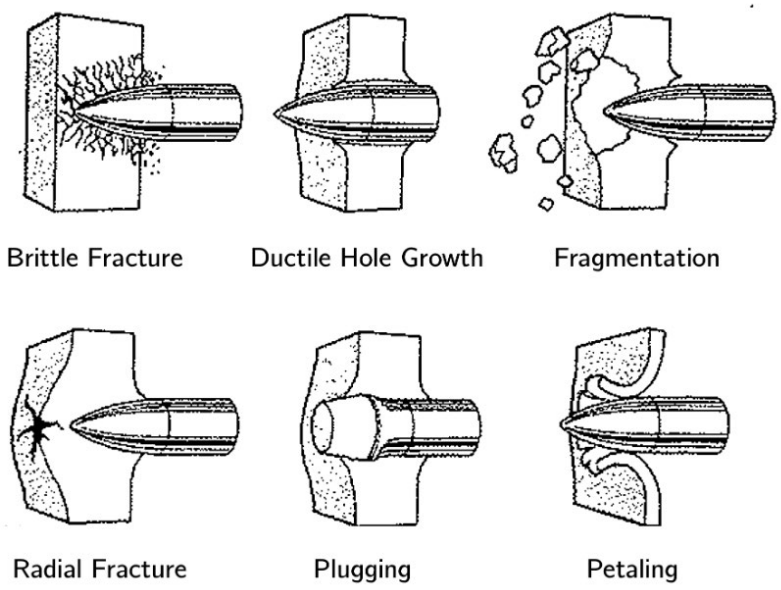


Figure 1.1: Common failure modes of targets subject to ballistic impact [Zukas, 1992]

Understanding the way a structure responds to ballistic impact is fundamental to the design of protective structures and armour systems in civil and military defence industries. Ballistic impact is the critical load case considered, and as such, energy absorption mechanisms and the perforation resistance capacity of structures has been a deep topic of study for decades [Backman and Goldsmith, 1978, Corbett et al., 1996, Brown, 1986]. Substantial effort has

been invested in order to physically understand and mathematically describe the phenomena taking place during ordnance ballistic penetration [Børvik et al., 1999a] and depending on the target material, geometry and impactor speed, perforation of a target can occur through a range of failure modes as shown in Figure 1.1. The objective of experimental testing is to determine the ballistic limit velocity, v_{bl}, the maximum velocity at which a target can resist perforation by a projectile. Measurements of projectile impact velocity, v_i, and residual velocity, v_r, can be made in controlled laboratory experiments whereby projectiles are fired at target plates at a range of velocities. To that end, a relationship known as the *initial versus residual velocity curve* can be established from v_i and v_r measurements in order to obtain v_{bl} [Børvik et al., 2009]. For the remainder of the thesis, this relationship will be referred to as the ballistic curve and is shown in more detail in Figure 1.2.

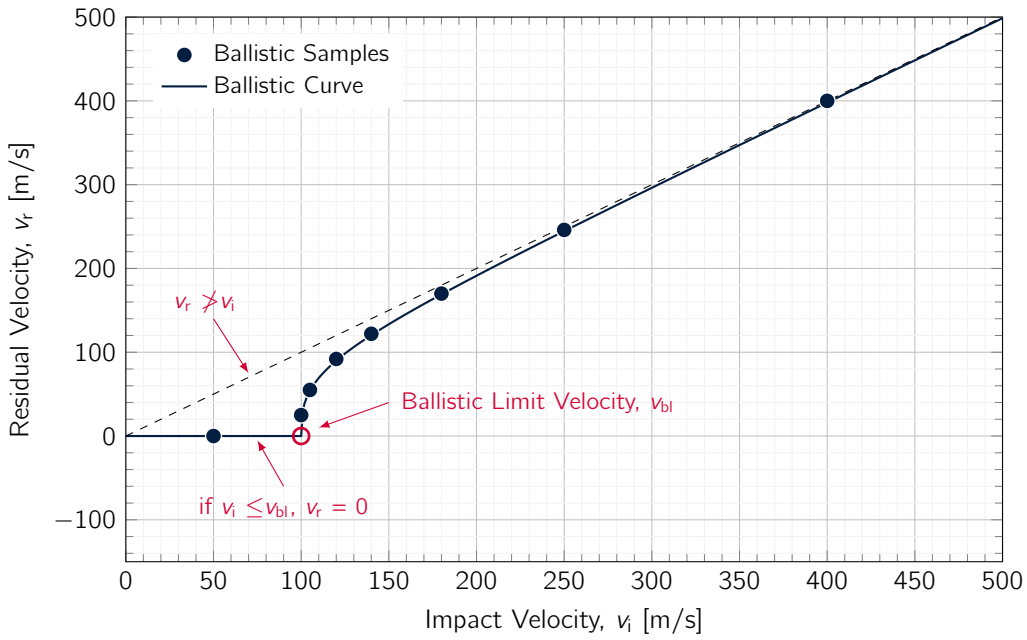


Figure 1.2: Diagram of the ballistic curve, indicating the location of the Ballistic Limit Velocity, v_{bl}.

The measured projectile impact velocity, v_i, and its corresponding residual velocity, v_r, are paired together to form a single ballistic sample, examples of which are marked on Figure 1.2. Given perfect experimental conditions, each ballistic sample across a complete range of impact velocities would lie upon the ballistic curve. In practice however, additional factors such as measurement error, angle of incidence of impact, local ductility and the homogeneity of the target material can introduce some variability into values recorded during experiments [Mohammad et al., 2020]. It is important to note that the ballistic curve is specific to the target material and the selected impactor (projectile). For example, the energy requirements to perforate a thicker target plate would be greater than that required to perforate a thin target

plate. To that end, it would be expected that the v_{bl} would increase for the thicker plate. Figure 1.2 also indicates two criterion of ballistic samples - the first states that the residual velocity cannot be greater than the impact velocity, hence $v_r \not> v_i$. A v_r greater than v_i would violate the law of conservation of energy as energy cannot be created or destroyed. It is physically impossible for a projectile to perforate a plate and gain kinetic energy without introducing further sources of energy to the system. In practice, upon perforation the kinetic energy from the projectile would be reduced and most of the excess transformed into heat energy and strain energy within the target plate to facilitate its deformation. The second criterion states that for impact velocities less than the ballistic limit, the residual velocity is 0 hence if $v_i \leq v_{bl}$, $v_r = 0$. This is an extension of the definition of the ballistic limit velocity which states that the v_{bl} is the minimum velocity at which a projectile can no longer perforate a target. For $v_i \leq v_{bl}$, the projectile does not possess the kinetic energy necessary to perforate the target plate and as such would rebound and damage the plate without perforation. Ballistic experiments published in the literature typically do not record the velocity of the rebounded projectile and label such occurrences with a v_r of 0 [Børvik et al., 1999a, Børvik et al., 2003, Børvik et al., 2005, Kristoffersen et al., 2021, Awerbuch and Bodner, 1974, Mohammad et al., 2020].

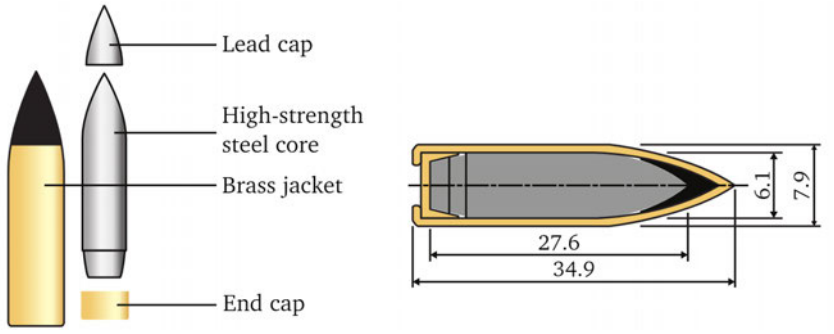


Figure 1.3: Specification of an ogive AP bullet and bullet core used in experimental tests [Holmen et al., 2015]

Ballistic testing encompasses a wide range of loading conditions, target materials and projectile variations. [Gupta et al., 2007] performed specific research considering the effect of impact velocity and target thickness on the deformation behaviour of aluminium plates impacted by blunt, ogive and hemispherical nosed steel projectiles. It was found that hemispherical nosed projectiles caused the highest global deformation of the target plates. Ogive-nosed projectiles were found to be the most efficient penetrator for the case of plate thicknesses between 0.5 and 1.5 mm whereas for thicknesses > 1.5 mm, the blunt nosed projectiles required the least energy to perforate the target plates. The v_{bl} of hemispherical projectiles was found to be the highest compared to the other two projectiles. A projectile commonly used in ballistic tests is an armour piercing (AP) 7.62 mm bullet shown in Figure 1.3 along with it's material composition and dimensions. It should be noted that in some numerical studies, tests are made with the hard steel component of the AP bullet only [Kristoffersen et al., 2020a] to simplify the modelling process. The extensive research of Børvik centres around studies of

the ballistic response of armoured steel and aluminium plates [Børvik et al., 1999a, Børvik et al., 2001b, Børvik et al., 2003, Børvik et al., 2005, Børvik et al., 2009, Børvik et al., 1999b]. The authors find that materials with high-strength properties demonstrate a clear link to an improved ballistic capacity, specifically a higher v_{bl}. However, ultra high-strength materials come with a compromise of the 'strength/ductility trade-off' which is a long-standing dilemma in materials science [Li et al., 2016a] that leads to a tendency for brittle fracture and fragmentation under impact. [Kristoffersen et al., 2020a] study was an example of innovation within experimental ballistic research that investigates Additive Manufacturing (AM) aluminium as a candidate for ballistic protection, which would allow structural designs to benefit from the flexible manufacturing advantages of using the AM methods. The combination of ultra-high strength and AM, however, introduces increased complexity and costs to ballistic experimental campaigns. This was demonstrated in an experimental study on AM maraging steel by [Costas et al., 2021]. The authors report significant fragmentation for both projectile and target in the ballistic tests. Often, the target plate was rendered unsuitable for further shots, reducing the total possible number of tests and thus increasing the cost of the experiment. Despite such challenges, the application of ultra high-strength AM materials presents an opportunity for areal density reduction of ballistic protection which is a key area of interest [Vemuri and Bhat, 2011]. Additional research is actively being pursued to optimise the strength versus ductility trade-off for AM maraging steel to improve the ballistic resistance capacity and subsequently make the materials better suited for use in the design of protective structures.

1.5 Analytical Modelling

Experimental tests are a vital requirement in the field of ballistic testing, especially for certification purposes. However, ballistic impact results are affected by many parameters, and there is a large number of phenomena involved. Thus, finding an optimal solution based mainly on experimental tests is not always feasible as it depends on the complexity of the required solution[Gregori et al., 2020]. As a result, alternative predictive modelling approaches have also become a rich topic of research. Analytical models are increasingly used for impact engineering given their inherent low-cost and ability to obtain initial estimates on the performance of ballistic protections. Analytical models can serve as a useful tool for the comparison of different materials and compositions. For example, [Chocron Benloulo and Sánchez-Gálvez, 1998] considered the ballistic impact against ceramic/composite armours and implemented a simple one-dimensional fully analytical model that obtained good correlation between analytical and experimental results. Their model allowed the calculation of residual velocity, residual mass, the projectile velocity and the deflection or strain histories of the backup material, variables that are important in describing the phenomenological process of penetration [Chocron Benloulo and Sánchez-Gálvez, 1998]. [Feli et al., 2011] improved upon that work by introducing a new energy formulation

for the composite backing based on yarns displacement and strain and subsequently [Bresciani et al., 2015] further developed the energy formulation based on the wave propagation theory. In addition, [Liaghat et al., 2013] developed a modified analytical model for the analysis of perforation into ceramic composite targets based on an expansion cavity model and [Naik et al., 2013] presented an analytical model for ballistic impact behaviour based upon wave theory and the energy balance between the kinetic energy of the projectile and the energy absorbed by different mechanisms. [hai Chen et al., 2017] considered the perforation of thin steel plates by hemispherical projectiles and proposed a dynamic analytical method that utilised the concept of plastic wave propagation based on rigid plastic assumptions. They found that their theoretical predictions were in good agreement with the experimental results in terms of both the radius of the bulging region and the residual velocity of the projectile when the strain rate effects of the target material were considered.

During experimental testing, the measured ballistic samples are fitted to a ballistic curve in order to obtain predictions for the v_{bl}. A common analytical model often used to fit the experimental results [Kristoffersen et al., 2021, Børvik et al., 2005, Børvik et al., 1999b] is the Lambert model [Ben-Dor et al., 2002], which is a generalisation of the Recht-Ipson (RI) model [Recht and Ipson, 1963] that describes v_r after complete perforation in relation to v_i and v_{bl} based on conservation laws of energy and momentum. The Lambert model reads

$$v_r = a\left(v_i^p - v_{bl}^p\right)^{\frac{1}{p}} \tag{1.1}$$

where a and p are material parameters adjusted to fit experimental data and estimate v_{bl} for a specific target. For tests on brittle materials, such as those by Costas et al. [Costas et al., 2021], large amounts of fragmentation can lead to values of a and p that violate conservation laws. In this case, Lambert model parameters a and p of values 1 and 2 respectively can be assumed, as for the RI model. The assumption made, however, is that no material is ejected during impact because the RI model is based on the ductile hole growth failure mode (see Figure 1.1). Analytical models in ballistics are therefore limited in scenarios with significant fragmentation. Where analytical models fall short, numerical modelling has been developed to provide a more comprehensive description of ballistic perforation mechanisms.

1.6 Numerical Modelling

Numerical modelling allows a wider description of the physics involved during impact phenomena but requires much greater efforts than analytical methods in order to perform the analysis. The Finite Element Method (FEM) is a widely used method for numerically solving differential equations arising in engineering and mathematical models and some of the earliest papers on FEM can be found in the works of [Schellbach, 1851] and [Courant, 1943]. The FEM is a general numerical method for solving partial differential equations in two or three space variables.

To solve a problem, the FEM subdivides a large system into smaller, simpler parts called finite elements. This is achieved by a particular space discretisation which is implemented by the construction of a mesh that encapsulates the object of study. The mesh defines the numerical domain for the solution which contains a finite number of points and the FEM formulation results in a system of algebraic equations. The FEM then approximates the unknown function over the domain [Reddy, 2018]. The simple equations that model these finite elements are then assembled into a larger system of equations that models the entire problem. Advancements in computing power in recent years have seen the use of numerical modelling with explicit Finite Element (FE) codes become commonplace to analyse the impact response and perforation processes over the last two decades. To model the impact response of a target accurately, the material behaviour must be captured as a function of high strain rates and temperatures [Børvik et al., 2001a, Dey et al., 2006]. The Johnson-Cook (JC) model [Johnson and Cook, 1983] is an example of a function regularly incorporated into numerical models to demonstrate such behaviour, and exists as a phenomenological relation that relies on experimentally defined parameters. Phenomenological models such as JC are often preferred to sophisticated methods that reflect physical micro-mechanical processes because the material constraints are far more readily available [Dey et al., 2007, Børvik et al., 2001a]. The literature contains a dense catalogue of numerical studies that consider the ballistic response of a wide variety of target materials and impactors.

For example, [Masri and Ryan, 2021] performed an in-depth numerical study on the perforation of multi-layered aluminium targets by rigid, nose-pointed projectiles using explicit FE code LS-Dyna. The model was used to make predictions regarding the ballistic limit and accompanied a new heuristic model that estimates the material resistance for each layer based on the specific cavitation concept. The behaviour of the of the aluminium alloy plate was described in the simulations via a linear equation of state and modified Johnson-Cook (MJC) constitutive model [Johnson and Cook, 1983]. The MJC has proved reliable in numerous studies on ballistic impact [Børvik et al., 1999b, Dey et al., 2007, Kristoffersen et al., 2020a, Børvik et al., 2009, Holmen et al., 2013, Børvik et al., 2005]. MJC is written as

$$\sigma_{\mathrm{p}} = \left(A + R(\epsilon_{\mathrm{p}})\right)\left(1 + \dot{\epsilon}^{*}\right)^{C}\left(1 - T^{*M}\right) \tag{1.2}$$

where σ_{p} and ϵ_{p} denote the equivalent stress and plastic strain respectively, A refers to the yield stress and C and M are material constants derived from experimental tests that govern strain rate sensitivity and thermal softening respectively. The normalised equivalent plastic strain rate $\dot{\epsilon}^{*}$ is defined by $\dot{\epsilon}^{*} = \dot{\epsilon}/\dot{\epsilon}_0$ where $\dot{\epsilon}$ and $\dot{\epsilon}_0$ are the strain rate and the reference strain rate respectively. The homologous temperature is described by $T^{*} = (T - T_r)/(T_m - T_r)$ where T_r is room temperature, T_m is the material melting temperature and T is actual material temperature. The actual temperature increment due to adiabatic heating is found by $\dot{T} = (\chi/\rho c_{\mathrm{s}}) \times \sigma\dot{\epsilon}$ where ρ is density, c_{s} is specific heat, and χ is the Taylor-Quinney coefficient, usually taken as 0.9 in the assumption that 90% of plastic work is dissipated as heat [Børvik et al., 2009]. The isotropic strain hardening term $R(\epsilon_{\mathrm{p}})$ is defined by the Voce hardening

rule $\sum_{i=1}^{2} R_i(\epsilon_p) = \sum_{i=1}^{2} Q_i\,(1 - \exp(-C_i\epsilon_p))$ where R_i are hardening terms which saturate at different levels of plastic strain governed by empirically determined parameters Q_i and C_i [Kristoffersen et al., 2020a]. When implemented into numerical models, MJC is often coupled with a criterion to allow fracture to occur in the material [Børvik et al., 2009]. [Dey et al., 2006] considered the accuracy of a one-parameter fracture criterion that can be defined via a single tensile test know as the Cockroft-Latham (CL) criterion [Kristoffersen et al., 2020a]. The CL criterion is defined as

$$D_c = \frac{1}{W_c} \int_0^{\epsilon} \langle \sigma_1 \rangle \mathrm{d}\epsilon \tag{1.3}$$

where σ_1 is the major principal stress, W_c is the CL failure parameter, and failure occurs when damage variable D_c equals unity. [Dey et al., 2006] found that an accurate, quantitative prediction of the ballistic limit velocity in steel target plates can be found using the CL criterion. The number of materials tests involved in characterisation is often cited as a factor in material model choice [Børvik et al., 2009, Dey et al., 2006]. Therefore, as a reliable, resource efficient method, the CL criterion is used alongside MJC in many studies involving impulsive loading for metallic materials [Kristoffersen et al., 2020a, Flores-Johnson et al., 2011, Basaran, 2017, Fras et al., 2019, Tria and Trebinski, 2015]. It should be noted however, that the CL criterion does not apply to the modelling of hard steel projectile core which is much more difficult to capture. To that end, numerical models can be simplified and computation time reduced by modelling the projectile core as a rigid body that remains intact - as performed by [Holmen et al., 2013]. However, this can result in an under-prediction of the ballistic limit velocity for high strength targets where damage to the projectile contributes to the effectiveness of the material as ballistic protection [Costas et al., 2021, Børvik et al., 2009].

Numerical studies in the literature however, are not limited to metallic targets. For example, [Krishnan et al., 2010] performed a numerical simulation of ceramic composite armour subjected to ballistic impact and also used FE code LS-Dyna. Due to the ceramic nature of the target plate, variants of the JC model are not applicable and instead the Johnson-Holmquist (JH) material model was used to model the impact phenomenon [Johnson and Holmquist, 1994]. Ceramic materials usually have high compressive strength but low tensile strength and tend to exhibit progressive damage under load due to the growth of microfractures. [Sabadin et al., 2018] numerically simulated and designed a SiC/Ultra High Molecular Weight polyethylene (UHMWPE) composite armour less than 20 mm thick that was able to resist a 7.62 mm AP projectile impact. Their advanced numerical model incorporated both mesh and mesh-less methods simultaneously and revealed good agreement between experimental and computational results both in terms of ballistic properties, deformations, fragmentation and fracture of the ballistic system. [Bresciani et al., 2016] investigated the fragmentation of a blunt shaped projectile made of a tungsten heavy alloy impacting against a ceramic Alumina tile. The ceramic tile was modelled such that the finite elements transform into Smooth Particle Hydrodynamics (SPH) elements when the failure criterion is met. [Gregori et al., 2020] created a numerical model studying the high-velocity impact of a 7.62 mm NATO Ball Projectile on multi-layer

alumina/aramid fibre composite ballistic shields. The model was developed in explicit code LS Dyna based on a full-Lagrangian FE analysis. Numerical studies have successfully been applied to a wide variety of loading scenarios and impact conditions in the literature. Finite methods provide a tool for evaluating the response of materials to impact and the failure modes of materials can be investigated specifically. Furthermore, simulations can be repeated and key design parameters can be modified to arrive at an optimal solution for the design of safe and fit for purpose structures. Such tools, however, require substantial amounts of computational resource depending on the loading case being studied. In addition, specialist knowledge and training is required to develop the models due to their complex nature and justifying the assumptions made to simplify the model. Finite methods suffer from discretisation errors and additional complexities arise with the application of different material models. Specifically, the main problem comes with obtaining and calibrating constitutive models as this usually requires large sets of experimental characterisation tests in conditions representative of the final application, which in most ballistic cases, are difficult to obtain in lab conditions.

1.7 Machine learning

Given the considerable computational cost associated with the increasingly sophisticated numerical models and boundary conditions, there has been an insurgence of ML research deployed within this field in order to improve the computational efficiency, and more importantly, enable the possibility of statistical analysis of the behaviour of materials across a large amount of input conditions [Yang et al., 2022]. One of the earlier ML studies to use artificial neural networks to predict the ballistic response of materials was by [Ryan and Thaler, 2013]. Specifically, the authors aimed to predict the perforation limits of spaced aluminium armour, Whipple shields, subjected to impact by aluminium projectiles at hypervelocity. The model utilised a multi-layer perceptron (MLP) architecture that, given a series of inputs that describe the mechanical properties of the projectile and target plate, predicted a binary output relating to whether or not perforation would occur. The results of which improved upon the previous empirical state of the art from 71% to 92%. Interestingly, this approach allowed the authors to perform a sensitivity analysis to determine which input parameters most heavily correlated with the output prediction. Whilst the majority of influential parameters aligned with the traditional parameters identified in the empirical calculations, some unexpected parameters were also identified [Ryan and Thaler, 2013].

The MLP has been a popular tool when applying machine learning techniques in the field of mechanics [Bobbili et al., 2020, Gao et al., 2019]. Liu et al. [Liu, 2003] used MLP in combination with a conjugate gradient method to optimize the design of functionally graded metal/ceramic materials. They were able to demonstrate that their trained neural network could effectively describe and handle the non-linearity between the design parameters and the objective

optimisation parameter, which in this case was the depth of penetration. In a separate study, [Bobbili et al., 2020] trained a MLP network to determine the residual velocity of a projectile impacting an aluminium 1100-H12 thin plate and demonstrated results from the model that were in agreement with corresponding experimental results. Motivated by such works, [Yang et al., 2022] considered a numerical approach that combines finite element modelling and machine learning to inform the material performance of an alumina ceramic tile undergoing high-velocity impact. They simulated the tile by incorporating a variation of the Johnson-Holmquist-Beissel (JHB) material model within the framework of SPH. The computational framework is used to simulate conditions by matching the results from both plate impact experiments and ballistic testing from the literature [Yang et al., 2022]. Once calibrated, the computational model is used to generate training data for the MLP to predict the residual velocity and projectile erosion for when the tile is impacted at high velocity. Similar to [Ryan and Thaler, 2013], a sensitivity analysis is performed to explore the effect of mechanical properties and impact simulation geometries on material performance. They demonstrated that a combined FEM and MLP approach is applicable to guide the structural-scale design of ceramic-based protection systems. [Teixeira-Dias et al., 2019] also used a MLP network in their study to predict the ballistic response of multi-layer armour plates. In this case, the authors train two MLP models, one of which is trained on the Lambert model described in Section 1.5 and another trained directly on experimental data available in the literature. This piece of work is presented in more detail in Chapter 3.

ML however, introduces a wider range of possibilities that have also been explored to facilitate our understanding of materials. [Pathan et al., 2019] used a supervised machine learning approach to predict the macroscopic stiffness and yield strength of unidirectional composites loaded in the transverse plane. The predictions were obtained without performing any physically-based calculations, but instead from image analysis of the material micro-structure supplemented with knowledge of the constitutive models for fibres and matrix. For each micro-structure, the corresponding homogenised, macroscopic mechanical properties were determined using FE simulations. Similar to [Yang et al., 2022], the computational models were used to create the training data for the ML models. In this case, the authors employed a gradient-boosted tree regression model that was trained using a 10-fold cross-validation training strategy utilising data from 1800 FE simulations. The model demonstrated that it could predict both elastic properties and yield strengths within a margin of error of 5 %. [Almustafa and Nehdi, 2022] also used a gradient-boosted tree regression model to predict the maximum displacement of reinforced concrete columns exposed to blast loading. The training set consisted of 420 test columns with thirteen features relevant to imperative column and blast properties (including dimensional properties of the RC column and its mechanical properties such as compressive strength and yield strength). The study was shown to provide accurate, generalised and stable predictions of maximum displacement via MAE, $MAPE$ and R^2 metrics of 3.63 mm, 13.31 % and 97.4 % respectively. Previously, the literature predicting the response of RC

columns to blasts consisted mainly of analytical or numerical models [Liu et al., 2019, Al-Bayti, 2017, Abladey and Braimah, 2014, Kadhom, 2016]. However, both approaches require an intimate understanding of the application to implement them successfully and the numerical methods in particular require additional modelling efforts and computational run time. Their ML approach eliminated the need for an in-depth understanding of the application, mitigated modelling complexity and effectively reduced computation time [Almustafa and Nehdi, 2022]..

Generative Adversarial Networks (GAN) in particular have shown remarkable results in modelling complex distributions [Grnarova et al., 2018]. [Aggarwal et al., 2021] conducted a detailed overview of the theory and applications of GANs in the literature and found that they have been applied successfully to a wide range of application types from 3D object generation [Chen et al., 2018, Ye et al., 2020, Yu et al., 2020], image processing [Zhang et al., 2020, Go et al., 2020], face detection [Jaiswal et al., 2020, Kowalski et al., 2020] and text transferring [Sixt et al., 2018]. The application of ML has been an active topic of research within the medical domain to identify chronic diseases [Battineni et al., 2020, Kaur et al., 2018] and GANs have proven to be a useful tool for some interesting applications. For example, for researchers developing robotic arms to support assisted living, they need to estimate the skeletons of hands that interact with objects from RGB images to facilitate their designs [Baek et al., 2020]. This topic of research is referred to as hand pose estimation and is complicated by severe occlusions and cluttered backgrounds. GANs have been successfully applied in this instance to help convert 2D images into a 3D hand pose estimation with remarkable success [Baek et al., 2020]. Their method retains state of the art performance, while eliminating the need to develop expensive fully annotated 3D skeletal models. Previously a complete and automatic pipeline for annotating 3D joint locations did not exist [Zimmermann et al., 2019]. GANs have also been used in the classification of brain tumor images [Oulbacha and Kadoury, 2020] and image segmentation [Yang et al., 2019].

GANs are also associated with the creation of synthetic healthcare data for AI projects to overcome related data-deficiency. For example, diverse data is critical for success when training deep learning models. Medical imaging data sets are often imbalanced as pathological findings are generally rare. In 2018, NVIDIA proposed a method to generate synthetic abnormal MRI images with brain tumours by training a GAN using two datasets available to the public of brain MRI imagery [Shin et al., 2018]. They demonstrated two key benefits from the synthetic data; the first is that they demonstrated improved performance on tumour segmentation by leveraging the synthetic images as a form of data augmentation. They utilised a GAN to supplement the available data to yield additional benefits from their ML model. The second is that they demonstrated comparable tumour segmentation results when trained on the synthetic data versus when trained on the real subject data. They declare that such results offer a potential solution to two of the largest challenges facing machine learning in medical imaging, namely the small incidence of pathological findings, and the restrictions around sharing patient data [Shin et al., 2018]. The availability of ballistic data suffers from similar issues of sparsity. Experimental

testing is a destructive process, which means that a comprehensive material characteristic campaign for the response of an armour plate across a range of impact velocities and material thicknesses incurs large costs rapidly. This cost is increased further when considering the response to different types of loading and projectile impact. There is therefore an opportunity for research to consider whether generative networks could be utilised to supplement sparse ballistic datasets and whether such methods can be used to augment material characterisation campaigns, without the need to perform additional destructive experiments or invest in the development of tailored numerical models. This thesis presents what, to the authors knowledge, is a novel implementation of generative networks in the field of ballistic impact, the details of which are presented in Chapters 4 and 5.

Chapter 2

Machine Learning

2.1 Preface

This chapter exists as a friendly introduction to ML and is not representative of the problems discussed later in the thesis. Nor is it intended as a contribution to knowledge or an advancement in the field of machine learning. It does however, serve several important purposes. The first is that it provides context and background information so that readers unfamiliar with artificial intelligence can understand some of it's key concepts, techniques and applications which are necessary to appreciate the more complicated use cases that follow. It is also intended to establish a common language. ML is a complex field with a rich vocabulary. This introduction chapter helps to establish that common language and forms a foundation of knowledge that can be built upon to understand the significance of the latter chapters in the context of the broader field of ML.

This chapter begins by introducing a brief history of AI and addressing some of the misconceptions between AI, ML and deep learning. It then introduces some important concepts such as neural networks, activation functions, loss functions and optimiser algorithms. GANs are a core component of this thesis and are also introduced. The chapter finishes by applying ML on a simple engineering problem by predicting the response of a single degree of freedom system using long-short term memory networks.

2.2 Introduction

AI is the capability of a computer system to mimic human cognitive functions such as learning or problem solving. Through AI, a computer system uses maths and logic to simulate the reasoning that people use to learn from new information and make decisions. AI has been claimed to offer transformational potential across a wide variety of sectors and industries [Collins et al., 2021]. Studies have reported that AI provides opportunities to reinvent business models [Duan et al., 2019], change the future of assembly work [Schwartz et al., 2019] and even enhance human capabilities [Dwivedi et al., 2021]. Traditionally, car manufacturing has been a rigid, automated process with steps executed by robots that execute pre-defined steps from procedural code. To improve flexibility, Mercedes-Benz for example, replaced some of those robots with AI-enabled collaborative robots, often referred to as cobots, and redesigned it's processes around human-machine collaborations [Daugherty, 2018]. The heightened interest in AI to transform economies is reflected in the scale of global spending - the International Data Corporation (IDC) predicts that global spending on AI will reach nearly 98 billion in 2023, which is more than double the 37.5 billion that was spent in 2019 [Majchrzak et al., 2016, Ransbotham et al., 2016]. [von Krogh, 2018] suggests a few reasons that attribute towards the recent increase in AI in recent years. Tremendous technical advancements have been made in some of the underlying AI methods such as current and conventional neural networks, many of which have been made open-source and regularly available to researchers [Collins et al., 2021]. In addition, the decreasing cost of computer hardware has contributed towards the reduction in computational expenditure. This has facilitated a notable increase in the speed at which research can take place and deeper models can now be trained on larger training sets than was previously possible.

Within the space of AI, the terms *Artificial Intelligence*, *Machine Learning* and *Deep Learning* are often used interchangeably and incorrectly - Figure 2.1 indicates the hierarchy of those terms. It is important to understand that AI is a general term that encompasses any work or research which enables a computer system to mimic human intelligence. ML however, is simply a subset of AI that defines the process of using mathematical models of data to help a computer learn without direct instruction. This enables a computer system to continue learning and improving on its own, based on experience. Historically, the term AI was first coined by [McCarthy et al., 2006] in the first academic conference on the subject - but the pursuit of intelligent machines began earlier than that as in 1945 [Bush, 2019] theorized a '*future device... in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory*'. In his essay he continues to state that *'...for years inventions have extended man's physical powers rather than the powers of his mind. Trip hammers that multiply the fists, microscopes that sharpen the eye, and engines of destruction and detection are new results, but not the end results of modern science'*. Such work exists as evidence of a shift in perspective

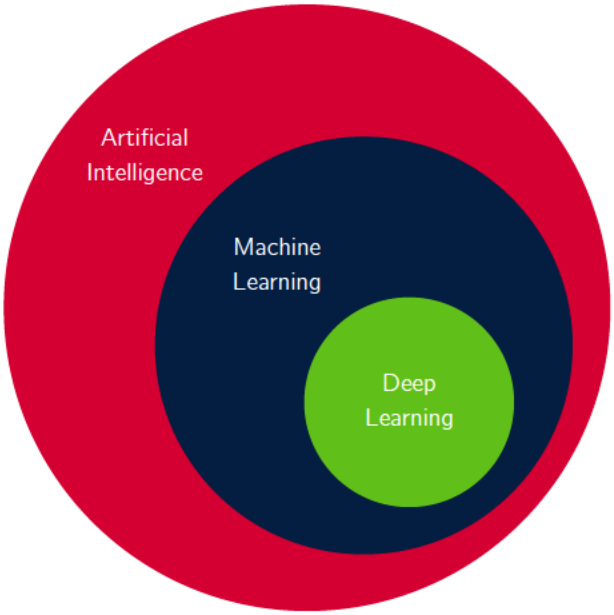


Figure 2.1: Euler diagram indicating the hierarchy of terms between artificial intelligence, machine learning and deep learning.

and acute awareness of the future possibilities of technology and intelligent machines. The earliest substantial work in the field of AI was completed in the mid 20th century by [Turing, 1950] who proposed a 'learning machine' that could learn and become artificially intelligent. Turing's conception is now known simply as the universal Turing machine of which the workings of all modern computers are based. Modern examples of AI which would not constitute as examples of ML include rule-based systems like chat-bots. Human-defined rules allow chat-bots to answer questions and are used by businesses to automate aspects of their customer service. No ML is required as the chat-bot receives its intelligence by a large amount of human input [Boden, 2014]. ML techniques however, provide an opportunity for computer systems to learn without direct instruction or human input. There are many types of ML techniques for different applications, but neural networks are a popular subset of ML and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. The basic building block of neural networks are artificial neurons, often referred to as perceptrons. The perceptron was first introduced by American Psychologist Frank Rosenblatt [Rosenblatt, 1958] and its design was heavily influenced by the biological neuron and its ability to learn. In their study, they summarise the biological theory on which the perceptron is built as follows:

1. The physical connections of the nervous system which are involved in learning and recognition are not identical from one organism to another. At birth, the construction of

the most important networks is largely random, subject to a minimum number of genetic constraints.

2. The original system of connected cells (neurons) is capable of a certain amount of plasticity; after a period of neural activity, the probability that a stimulus applied to one set of cells will cause a response in some other set is likely to change due to some relatively long-lasting changes in the neurons themselves.

3. Through exposure to a large sample of stimuli, those which are most 'similar' (in some sense that must be defined in terms of the particular physical system) will tend to form pathways to the same sets of responding cells. Those which are marked 'dissimilar' will tend to develop connections to different sets of responding cells.

4. The application of positive and/or negative reinforcement (or stimuli which serve this function) may facilitate or hinder whatever formation of connections is currently in progress.

5. *Similarity*, in such a system, is represented at some level of the nervous system by a tendency of similar stimuli to activate the same set of cells. Similarity is not a necessary attribute of a particular formal or geometrical classes of stimuli, but depends on the physical organisation of the perceiving system, an organisation which evolves through interaction with a given environment.

Despite being published in 1958, the points raised regarding the biological theory are still analogous to the way in which artificial neural networks learn and are utilised today. This section explores neural networks in more detail and explains how perceptron networks are organised and optimised such that significance can be applied to certain inputs to facilitate learning.

2.2.1 The Perceptron

Figure 2.2 shows a diagram of a perceptron, or node of an Artificial Neural Network (ANN), in its simplest form.

The perceptron works by adding together the product of each input x_i multiplied by its associated weight w_i to form a weighted sum. To that end, the output y given by this node can be expressed as

$$y = x_1 w_1 + x_2 w_2 + x_3 w_3 + \cdots + x_n w_n \tag{2.1}$$

The output y is therefore a product of its inputs and weights and as such, the larger the weight w_i the greater the influence that input x_i has on the output. This is an important concept of machine learning and during training, the goal is to optimise the weight values such that

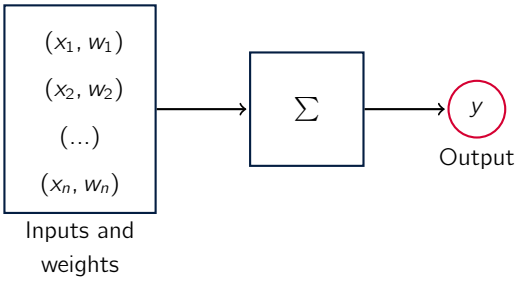


Figure 2.2: Schematic of a simple perceptron.

significance can be applied to the correct inputs to modify the output y such that it is closer to the target/desired value. Perceptrons used in neural networks expand upon this basic principle but include two additional components known as the bias b and the activation function F. b is an additional trainable parameter that consists of a numeric value that is added to the weighted sum of the inputs and F performs a final transformation in order to help the network learn complex patterns in the data. Activation functions are discussed in more detail in Section 2.3 and a schematic of a typical perceptron is shown in Figure 2.3. To that end, the operations

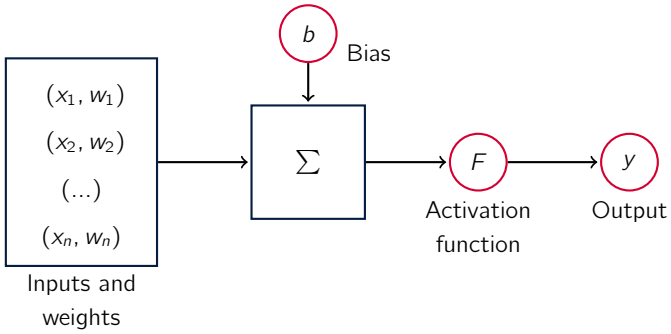


Figure 2.3: Schematic of a perceptron node.

that occur at a single node within a neural network can be described mathematically as

$$y = F\left(b + \sum_{i=1}^{n} w_i x_i\right) \tag{2.2}$$

where n refers to the total number of inputs entering the node. On its own however, a single perceptron is trivial in the sense that complex operations can be performed when these nodes are combined and arranged into layers to create a mesh-like network. This network is typically referred to as a Multi Layer Perceptron (MLP). It is composed of an input layer that receives the signal, an output layer that makes a decision or prediction regarding the input, and the hidden layers that act as the computational engine of the MLP. These networks are often applied to supervised learning problems where they train on a set of input-output pairs and learn

to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters of the network such that the discrepancy between the predicted value and expected value is minimised. Once a model is trained, it is possible to provide it with new inputs and make new predictions based on what the model has learned during training.

The term deep learning is given to networks composed of multiple hidden layers. A hidden layer is a vector of perceptrons that switch on or off as the input is fed through the network. Each layer's output becomes the subsequent layers' input, stemming from the initial input layer that receives the data that was fed into the network. Pairing the model's adjustable weights with input features makes it possible to assign significance to those features with regard to how the network classifies and clusters input. Deep learning networks are distinguished from single hidden-layer neural networks by their depth, that is, the number of node layers through which data must pass through before reaching the output. The mathematical reasoning behind why layering is useful is denoted by Taylor's theorem where any function can be represented as an infinite linear combination of polynomials. This is analogous to a layered network and thus with an infinitely large network it is possible to model any function precisely [Goodfellow et al., 2016].

Figure 2.4 compares a single layer MLP network with that of a deep MLP network. A MLP network is structured such that key parameters including the number of inputs, outputs, number of hidden layers and the type of activation functions that are selected are best suited to model the problem case. MLP are well suited to regression-based prediction problems where a real quantity is desired given a set of inputs. The network is trained on a tabular data set arranged in samples, where a single sample consists of a series of inputs and its corresponding target output(s).

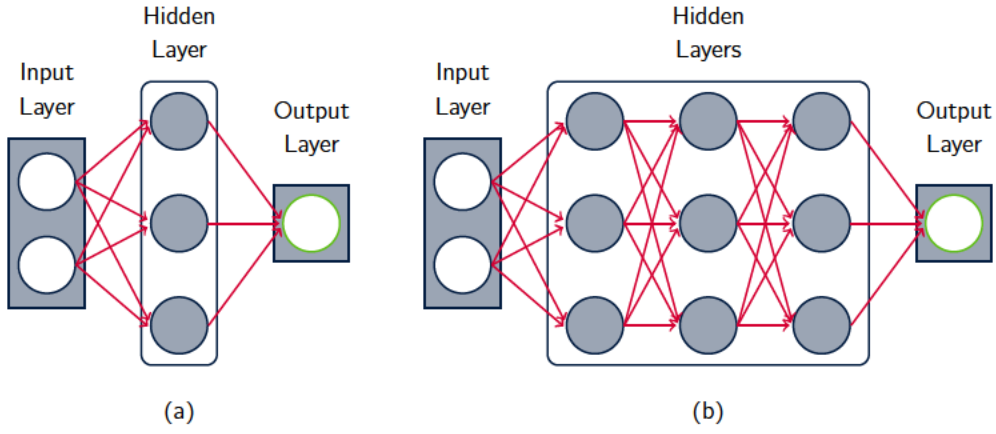


Figure 2.4: Schematic diagram of (a) a MLP network and (b) a deep MLP network.

2.3 Activation Functions

2.3.1 Overview

Activation functions play an important role within neural networks and introduce non-linearity into the output of a perceptron. An activation function is a function added into active nodes of an Artificial Neural Networks (ANN) in order to help the network learn complex patterns in the data. Figure 2.5 indicates the position of the activation function within an active node.

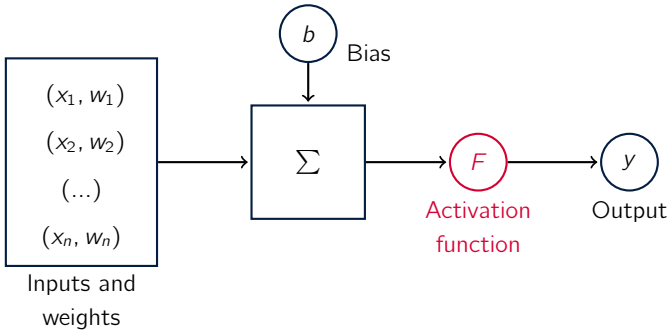


Figure 2.5: Location of activation function within an active node.

The activation function takes the combined sum of the inputs and weights and moderates it before the output is passed on to subsequent nodes. To this end, the activation function takes the following input

$$b + \sum_{i=1}^{n} w_i x_i \tag{2.3}$$

where w_i is a weight associated with a given input x_i, n is the number of inputs and b is the bias added to the combined sum of inputs and weights. The activation function is used to determine the extent to which the signal progresses through the network. It converts the signal it receives through a form of gradient processing into a new form that can be taken as input by the next cell. It should be noted that the terms 'activation function' and 'transfer function' are often used interchangeably in the literature. An activation function can be either linear or non-linear depending on the function it represents and is used to moderate the output of neural networks across a wide range of domains from object recognition and classification [Krizhevsky et al., 2017, Szegedy et al., 2014], to speech recognition [Sainath et al., 2015, Graves et al., 2013], cancer detection systems [Albarqouni et al., 2016, Wang et al., 2016, Cruz-Roa et al., 2013], self-driving cars [Uçar et al., 2017, Chen et al., 2015] and many others, with [Olgac and Karlik, 2011] stating that a proper choice of activation function improves results in neural network computing.

A linear model refers to the linear mapping of an input function to an output, such as the final prediction between class label and prediction in a classification network, and is given by the affine transformation in most cases [Goodfellow et al., 2016]. The affine transformation is defined as any transformation that preserves co-linearity (i.e. all points lying on a line initially still lie on a line after transformation) and ratios of distances (i.e. the midpoint of a line segment remains the midpoint after transformation). Furthermore, the neural networks produce linear results from the mapping of $b + \sum_{i=1}^{n} w_i x_i$ and the output of these nodes can be written as

$$y = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b \tag{2.4}$$

The outputs of each node in each layer are fed into the subsequent layer for multilayered networks until the final output is obtained. In this form, however, they remain linear. In order to learn patterns in the data, it is important to convert these linear inputs into non-linear output for further computation and as such the need for the activation function arises. The activation functions can be applied to the outputs of the linear models to produce the transformed non-linear outputs. The non-linear output after the application of the activation function can therefore be written as

$$y = F(w_1 x_1 + w_2 x_2 + .. + w_n x_n + b) \tag{2.5}$$

where F is the activation function. Note that this is equivalent to Equation 2.2. Converting linear signals to non-linear signals allows the neural network to capture the learning of high order polynomials beyond one degree for deeper networks. A special property of the non-linear activation function is that they are differentiable else they cannot work during backpropagation of the deep neural networks [Goodfellow et al., 2016]. Backpropagation is a crucial part of the training process and used to update the weights and bias values at each node in the network.

2.3.2 Sigmoid and tanh Activation Functions

The simplest activation function is known as the linear activation where no transform is applied at all. A network that contains primarily linear activation functions is very simple to train, but cannot learn complex mapping functions. Non-linear activations are preferred as they allow for optimisation such that the weights and biases in each node can learn more complex structures within the training data [LeCun et al., 2015]. Traditionally, two widely used non-linear activation functions are the sigmoid and hyperbolic tangent (tanh) functions and are shown in Figure 2.6.

The sigmoid is a non-linear activation function used mostly in feedforward neural networks. It is a bounded differentiable real function, defined for real input values, with positive derivatives everywhere and some degree of smoothness [Han and Moraga, 1995]. The sigmoid function is given by the relationship

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{2.6}$$

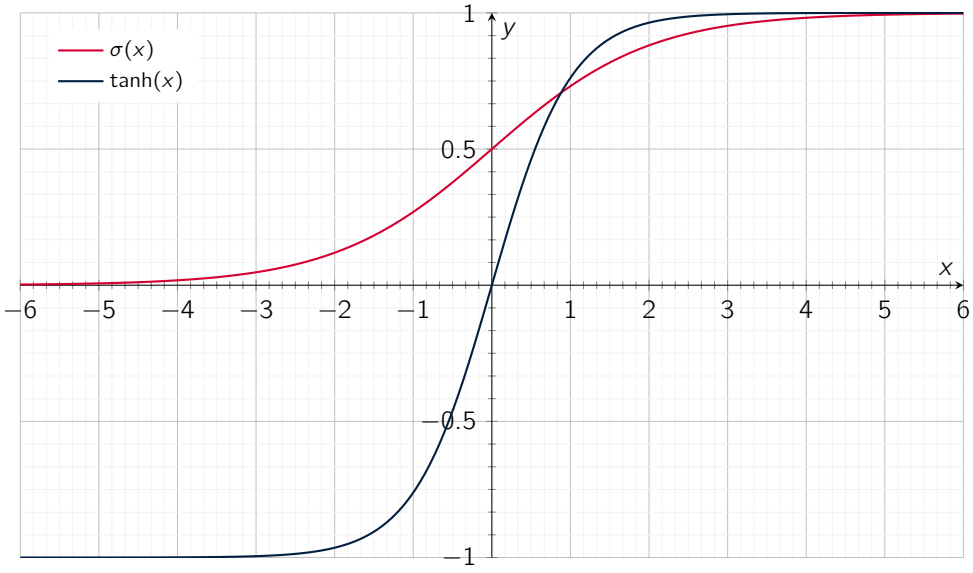


Figure 2.6: sigmoid σ and tanh activation functions for real values of x between -6 and 6.

The sigmoid function often appears in the output layers of DL architectures as for any given x the function returns a value between 0 and 1. In light of this, they are well suited for use in predicting probability-based output and are regularly applied in binary classification problems. However, the sigmoid activation function suffers from major drawbacks, which include sharp damp gradients during backpropagation from deeper hidden layers to the input layers, gradient saturation, slow convergence and non-zero centered output, causing the gradient updates to propagate in different directions [Nwankpa et al., 2018].

In order to address some of the shortcomings associated with the sigmoid function, other forms of activation function were proposed such as the hyperbolic tangent function known as tanh [Nwankpa et al., 2018]. Tanh is a smoother, zero-centered function that for any given x returns a value between -1 and 1. The tanh function is defined by the following relation:

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

The tanh activation function is preferred over the sigmoid function as it demonstrates improved training performance for multi-layer neural networks [Olgac and Karlik, 2011, Neal, 1992]. The main advantage provided by this function is that it produces a zero centered output and thereby aids the backpropagation process. However, like the sigmoid function, the tanh function also suffers from the vanishing gradient issue. This is because both functions squash their input into a very small output range in a very non-linear fashion. For example, for all values of x, sigmoid maps the output to a small range between 0 and 1. However, for most positive values of x when $x > 5$ the output is very close to 1 and for most negative values of x when $x < -5$ the

output is close to 0. This issue is referred to as saturation [LeCun et al., 2015] and, in practice, it means that there are large regions of the input space which are mapped to an extremely small range. In these regions of the input space, even a large change in the input will produce a small change in the output. Hence the gradient is small and the parameters of the network will be updated less vigorously. This issue is amplified when multiple layers that include such non-linearities are stacked. For instance, the first layer within the network will map a large input region to a small output region, which will subsequently be mapped to a smaller output region in the next layer. This trend will continue such that even a large change in the parameters of the first layer will have a reduced impact on the output. It is possible to negotiate the issue of vanishing gradients by using activation functions that do not have the property of squashing the input space into finer boundaries. A popular choice is the Rectified Linear Unit (ReLU) activation function.

2.3.3 Rectified Linear Unit Activation Functions

Stochastic gradient descent and backpropagation are two important concepts when training neural networks that are discussed in more detail in Section 2.4. However, in order to utilise them and train deep neural networks with most success, an activation function is required that behaves like a linear function, but is in fact non-linear such that complex relationships in the data can be learned. The ReLU satisfies these requirements as it performs a threshold operation to each input element where the function is linear for values of $x > 0$, yet it is a non-linear function as negative values are output as 0. As such, the ReLU can be defined as:

$$f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases} \tag{2.8}$$

The ReLU represents a nearly linear function and therefore preserves the properties of linear models that made them easy to optimise and converge faster with gradient-descent methods [Goodfellow et al., 2016]. An advantage of using the ReLU units is that they enhance the speed of computation since it is not required to compute divisions or exponentials. Specifically, [Krizhevsky et al., 2017] found that the speed of convergence increased by a factor of 6 when using ReLUs compared to the sigmoid and tanh functions. However, the ReLU has a limitation that it easily overfits compared to the sigmoid function, although techniques including dropout have been adopted to reduce the effects of over-fitting of ReLUs and the rectified networks have been shown to improve the performances of the deep neural networks [Glorot and Bengio, 2010]. The ReLU and its variants are used regularly in different architectures of deep learning due to their simplicity and reliability — applications include restricted Boltzmann machines [Nair and Hinton, 2010] and many convolutional neural network architectures [Krizhevsky et al., 2017, Szegedy et al., 2014, Xu et al., 2015]. However, ReLU units can be fragile during training and cease to produce useful outputs. For example, a large gradient flowing through a ReLU

could cause the weights to update in such a way that the neuron will not activate on any data point again. If this happens, the gradient passing through the unit will be zero from that point on. In this sense, a dead ReLU unit is one that will always output the same value for any input and will no longer contribute to any form of learning. This phenomenon can be emphasised if the learning rate is set too high as the increased learning rate can destabilise the already fragile rectified units. Reducing the learning rate reduces the likelihood of dead neurons, however a modified ReLU known as the Leaky ReLU (LReLU) is an alternative proposal to address the issue.

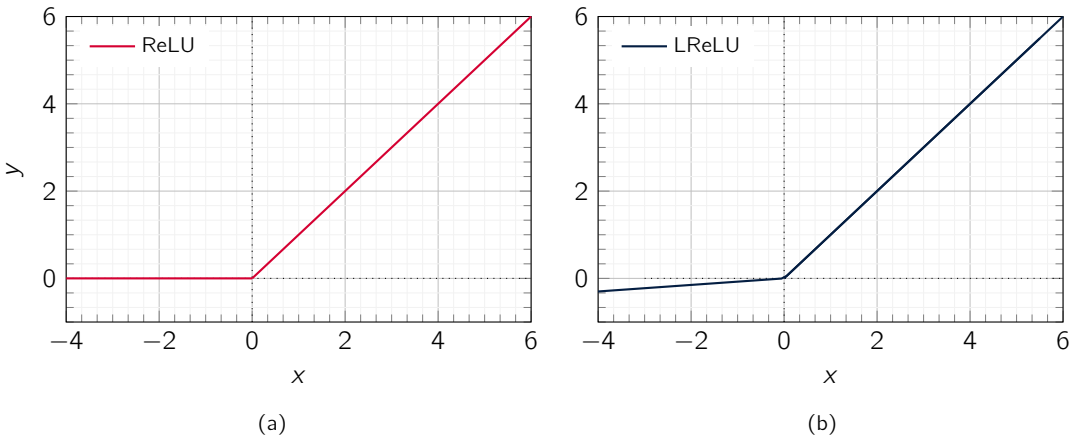


Figure 2.7: (a) ReLU activation function and (b) LReLU activation function across a range of real values of x from -4 to 6.

Figure 2.7 shows the output of both the ReLU and LReLU activation functions for real values $-4 < x < 6$. The LReLU was first proposed in 2013 as an activation function that introduces a small negative slope to the ReLU to sustain and keep the weight updates alive throughout the entire propagation process [Maas et al., 2013]. The LReLU function introduces the α parameter to address the issue of dead neurons and assures that the gradients will not be zero during training. The LReLU computes the gradient with a very small constant value for the negative gradient α, typically in the range of 0.01 to 0.02. As such, the LReLU activation function can be described as

$$f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ \alpha x, & \text{if } x_i < 0 \end{cases} \tag{2.9}$$

The LReLU function therefore returns an identical result when compared to the standard ReLU for positive values of x. To this end, there is no significant result improvement when compared to the traditional ReLU and tanh functions except in sparsity and dispersion. Further adaptations of the ReLU include Parametric ReLUs (PReLU), where the negative part of the

function learns adaptively whilst the positive region remains linear, and Randomized Leaky Relu (RLReLU) which is a dynamic variant of the LReLU where a random number sampled from a uniform distribution is used to train the network.

2.3.4 Softmax Activation Function

As mentioned previously, activation functions are implemented in neural networks to moderate the output from its layers. Softmax activation functions, or *softargmax* as they are sometimes referred to [Goodfellow et al., 2016], are often used as the final activation function of a neural network to normalise the output of a network to a probability distribution over predicted output classes. Softmax is a mathematical function that converts a vector of numbers it receives into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. Classification networks, are configured to output n values, where n is the number of classes in the classification task. Examples of classification networks trained on popular MNIST and CIFAR10 datasets are presented in Section 2.5.1. The softmax function is used to normalise those outputs and converts them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class. Mathematically, softmax is defined as

$$S_i(y) = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \tag{2.10}$$

where y is an input vector to the softmax function S that consists of n elements for n classes. The subscript i denotes the i-th element of the input vector and accepts any numeric value. $\sum_{j=1}^{n} e^{y_j}$ is a normalisation term that ensures the values of output vector $S_i(y)$ sums to 1 and that each element is in the range 0 and 1 thus ensuring a valid probability distribution. Consider a classification network trained on the CIFAR10 dataset that aims to classify input images into one of 10 classes. The last fully connected layer of the network outputs a vector of logits, L_x, that is passed through a softmax layer that transforms the logits into normalised probabilities, P. Logits, in this instance, refers to the vector of raw (non-normalised) predictions generated by the classification model. The probabilities P correspond to the model's prediction of the likelihood that the input image belongs to each of the 10 possible classes. This idea is illustrated in Figure 2.8 and considers an aeroplane as an input image. It can be seen that the value corresponding to the aeroplane class in probabilities P returns the largest value of 0.79. The model therefore believes that the input image belongs to the class of *aeroplane*.

In summary, this section highlighted a few of the common activation functions used in the literature. In practice, however, there are many more functions used and with each function there are often variants used to accommodate different applications. A summary of the main activation functions used across various domains is shown in Table 2.1.

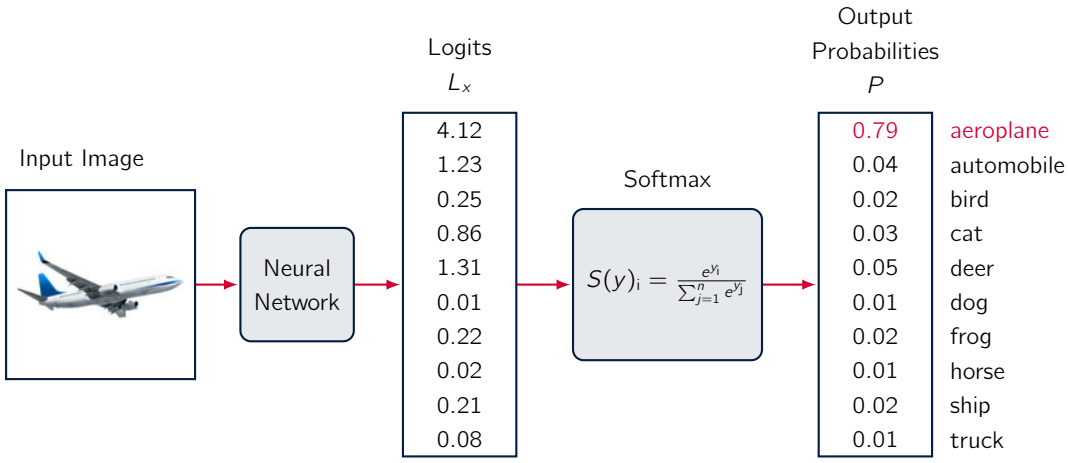


Figure 2.8: Schematic of softmax functionality within a classification network featuring 10 classes that correspond to the classes within the CIFAR10 dataset [Krizhevsky and Hinton, 2009].

Table 2.1: Deep learning activation functions and their corresponding equations for computation.

Function	Computation Equation	Reference
Sigmoid	$f(x) = \frac{1}{(1+e^{-x})}$	[Han and Moraga, 1995]
HardSigmoid	$f(x) = \max\left(0, \min\left(1, \frac{(x+1)}{2}\right)\right)$	[Nwankpa et al., 2018]
SiLU	$f(x) = z_i\alpha(z_i)$	[Elfwing et al., 2017]
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	[Nwankpa et al., 2018]
Hardtanh	$f(x) = \begin{cases} -1, & \text{if } x < -1 \\ x, & \text{if } -1 = x \leq 1 \\ 1, & \text{if } x > 1 \end{cases}$	[Nwankpa et al., 2018]
Softmax	$f(x_i) = \frac{e(x_i)}{\sum_j e(x_j)}$	[Nwankpa et al., 2018]
ReLU	$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$	[Nair and Hinton, 2010]
LReLU	$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x_i < 0 \end{cases}$	[Maas, 2013]
PReLU	$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x \leq 0. \end{cases}$	[He et al., 2015a]
RReLU	$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases}$	[Xu et al., 2015]

2.4 Optimisers and Loss Functions

Whenever a neural network is trained, the trainable parameters of the network, such as the weights and biases of active nodes, are optimised with the goal of changing the next output such that it is closer to the target value. This is performed by an optimisation algorithm that is used to find the values of parameters (coefficients) that minimises a cost function, L, and is often also referred to as the loss function or error function [Goodfellow et al., 2016]. In machine learning, cost functions are used to estimate how badly a model is performing. A cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between input data $\mathbf{x}$ and target data $\mathbf{Y}$. This is is typically expressed as a finite value, termed loss, in reference to a difference or a distance between the predicted value and the expected value; therefore a larger value indicates a worse performing model. The cost function is evaluated during training by running the model iteratively to compare the estimated predictions against the 'ground truth', i.e. the target values of $\mathbf{Y}$ in the training set. To that end, the objective of a ML model therefore is to find parameters, weights, or an architecture that minimises the cost function. If the cost function returns a very small value, this indicates little difference between the predictions made by the model and the target values, and is evidence that the network has been able to learn features from the data within the training set successfully.

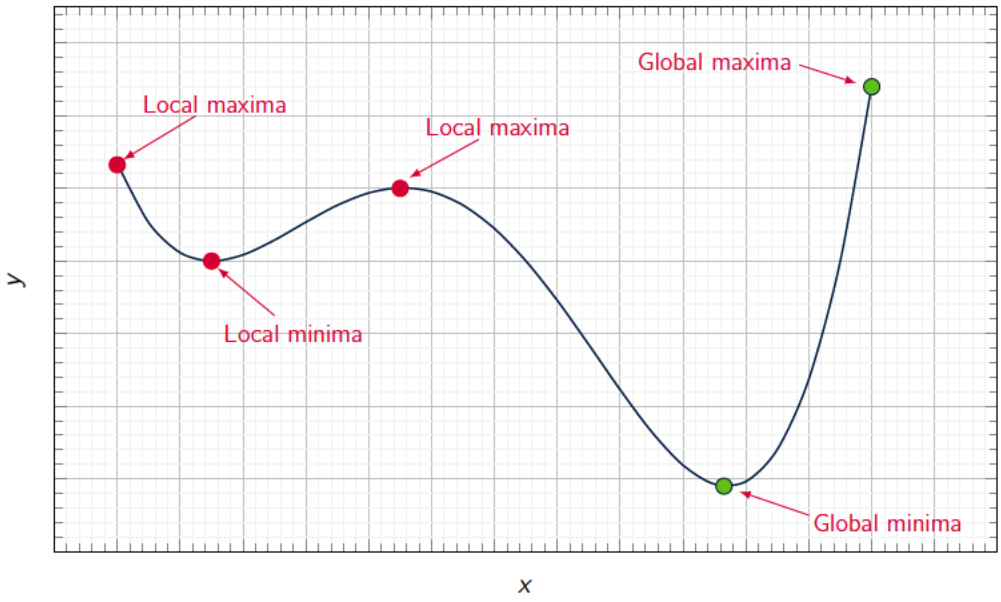


Figure 2.9: An example of an arbitrary non-convex cost function highlighting the locations of local and global minima and maxima.

Cost functions, however, are non-convex and do not have a single minimum value, but instead may possess many local minima and maxima as shown in Figure 2.9. Therefore, to

minimise the loss from a neural network, an algorithm known as backpropagation is used [Goodfellow et al., 2016]. Backpropagation calculates the derivative of the cost function with respect to the parameters in the neural network. It uses this information to determine the direction in which to update the parameters to improve the performance of the model. This direction is more commonly referred to as the neural networks gradient, and a steeper gradient will result in a larger change to its parameters. However, before updating the model with the gradient, it is first multiplied by an additional parameter known as the learning rate. The learning rate is a parameter that is often defined prior to training the network and common learning rate values lie between 0.01 and 0.2 for a range of applications [Smith, 2015]. If the learning rate is too high, the combination of the learning rate multiplied by the gradient might result in the output from the network over-stepping the minimum, resulting in a model that is not as successful as it could have been. Conversely, if the learning rate is too low the optimisation process is very slow. This is because the combination of the learning rate and gradient will result in a very small value, and as such fine updates will be made to the trainable parameters of the network. This means that more training iterations will be required for the algorithm to minimise the loss function than might be necessary. In addition, the algorithm may converge to an unfavourable local minimum, resulting in a model that has stabilised about a sub-optimal point. This is where optimisers add value to the training process, many optimisers calculate the learning rate automatically but are responsible for applying the gradients to the neural network to facilitate learning. A good optimiser is one that both trains a model quickly and prevents convergence at sub-optimal local minima.

2.4.1 Optimiser algorithms

This section presents the optimiser algorithms that are candidates for use in the studies that constitute the main body of this thesis. A brief overview of the Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSProp), AdaGrad and Adam optimisation algorithms is given first before evaluating their performance on a classification task on the popular MNIST [Lecun et al., 1998] and CIFAR10 [Krizhevsky and Hinton, 2009] datasets.

Stochastic Gradient Descent

SGD and its variants are among the most common optimisation algorithms used for machine learning applications in general and deep learning in particular. It is possible to obtain an unbiased estimate of the gradient by computing the average gradient on a minibatch of m examples drawn from independent and identically distributed random variables from the data-generating distribution [Goodfellow et al., 2016], i.e. a random selection of examples from the training set. SGD updates the trainable network parameters, such as the weights and biases $\boldsymbol{\theta}$, to minimise

the loss function by taking small steps at each iteration in the direction of the negative gradient. The SGD algorithm is presented in Table 2.2.

Table 2.2: SGD Optimisation algorithm [Goodfellow et al., 2016].

SGD Algorithm Stochastic gradient descent (SGD) update

Require: Learning rate schedule ϵ_1, ϵ_2, ...
Require: Initialise Parameter $\boldsymbol{\theta}$

start
  $k \leftarrow 1$
  while stopping criteria not met **do**
    Sample a minibatch m examples from the training set $\mathbf{X}$, $\{\mathbf{X}_1, ..., \mathbf{X}_m\}$, with corresponding targets $\mathbf{Y}_i$
    Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{X}_i; \boldsymbol{\theta}), \mathbf{Y}_i)$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \mathbf{g}$
    $k \leftarrow k + 1$
  end while

Where L is the loss function elected for the optimisation. A crucial parameter for the SGD algorithm is the learning rate ϵ. SGD is often described as using a fixed learning rate, however in practice it is necessary to gradually decrease the learning rate over time hence the learning rate at iteration k is denoted as ϵk. This is because the SGD gradient estimator introduces a source of noise (via the random sampling of m training examples) that does not vanish once a minimum has been reached [Goodfellow et al., 2016]. By comparison, the true gradient of the total cost function becomes small and then 0 as a minimum is approached and ultimately met by using batch gradient descent, hence with batch gradient descent it is possible to use a fixed learning rate.

A common issue with the SGD algorithm is that it can oscillate along the path of steepest descent towards the optimum which results in some instability. An additional momentum term can be added to the original SGD algorithm to reduce the oscillation which is designed to accelerate learning, particularly in the face of high curvature, small but consistent gradients or noisy gradients [Polyak, 1964, Murphy, 2012]. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. Formally, the momentum algorithm is introduced via a new variable $\boldsymbol{\gamma}$ that plays the role of velocity, it is a vector that refers to the direction and speed at which the trainable parameters move through parameter space. An additional momentum hyperparameter $\xi \in [0, 1]$ determines the speed at which the contributions of previous gradients exponentially decay and

the update rule is given by

$$\mathbf{v} \leftarrow \xi\boldsymbol{\gamma} - \epsilon\nabla_{\boldsymbol{\theta}}\left(\frac{1}{m}\sum_{i=1}^{m} L(f(\mathbf{x}_i;\boldsymbol{\xi}),\mathbf{y}_i)\right), \tag{2.11}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\gamma}. \tag{2.12}$$

The larger ξ is relative to ϵ, the more previous gradients affect the current direction. The updated SGD algorithm that includes momentum is given in Table 2.3.

Table 2.3: SGD Optimisation algorithm.

SGD Algorithm Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter ξ
Require: Initialise Parameter $\boldsymbol{\theta}$, initial velocity (momentum) $\boldsymbol{\gamma}$
 	
start
$k \leftarrow 1$
while stopping criteria not met **do**
Sample a minibatch m examples from the training set $\mathbf{X}$, $\{\mathbf{X}_1,...,\mathbf{X}_m\}$, with corresponding targets $\mathbf{Y}_i$
Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\mathbf{X}_i;\boldsymbol{\theta}),\mathbf{Y}_i)$
Compute velocity update: $\boldsymbol{\gamma} \leftarrow \xi\boldsymbol{\gamma} - \epsilon\mathbf{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\gamma}$
$k \leftarrow k + 1$
end while
end

The learning rate is one of the most difficult hyperparameters for neural network researchers to set as it significantly affects the performance of the model. The momentum algorithm helps mitigate this issue, however it does so at the expense of introducing another hyperparameter that must be defined prior to training. AdaGrad, RMSProp and Adam optimisation algorithms are examples of incremental methods that directly adapt the learning rates of specific model parameters.

AdaGrad

The AdaGrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of the historical squared values of the gradient [Duchi et al., 2011]. The trainable parameters $\boldsymbol{\theta}$ with the largest partial

derivative of the loss have a correspondingly rapid decrease in their respective learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. The AdaGrad optimisation algorithm is presented in Table 2.4.

Table 2.4: AdaGrad Optimisation algorithm [Goodfellow et al., 2016].

AdaGrad Algorithm The AdaGrad optimisation algorithm
Require: Global learning rate ϵ
Require: Initialise Parameter $\boldsymbol{\theta}$
Require: Small constant δ to stabilise division by small numbers
start
Initialise gradient accumulation variable $\mathbf{r} = 0$
while stopping criteria not met **do**
Sample a minibatch m examples from the training set $\mathbf{X}$, $\{\mathbf{X}_1, ..., \mathbf{X}_m\}$, with corresponding targets $\mathbf{Y}_i$
Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{X}_i; \boldsymbol{\theta}), \mathbf{Y}_i)$
Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$
Compute update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while
end

It should be noted that the mathematical operator $\odot$ denotes the Hadamard Product, which for ML applications refers to the component-wise multiplication of matrices [Horn and Johnson, 1985]. δ is a small constant added to prevent division by zero and improve stability, often this value is set to 10^{-7} [Goodfellow et al., 2016]. The AdaGrad algorithm individually adapts the learning rate based on the accumulation of gradients, as such, an additional vector $\mathbf{r}$ is defined and initialised at 0 for all parameters $\boldsymbol{\theta}$. AdaGrad demonstrates desirable properties when applied to convex optimisation problems. Empirically, however, for training deep learning models, the accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate [Goodfellow et al., 2016]. As a result, AdaGrad performs well for some but not all deep learning applications.

Root Mean Square Propagation (RMSProp)

The RMSProp algorithm was first proposed by Geoffrey Hinton and exists as a modification of AdaGrad for improved performance on non-convex optimisation problems by changing the

gradient accumulation into an exponentially weighted moving average [Hinton, 2012]. The AdaGrad algorithm was designed to converge rapidly when applied to a convex function and when it is applied to a non-convex function to train a neural network, the learning trajectory may pass through different structures and eventually arrive at a region that is a locally convex minima. AdaGrad shrinks the learning rate according to the entire history of the squared gradient and as a result, the learning rate may be too small before arriving at such a convex structure. Instead, RMSProp uses an exponentially decaying average to discard history from the extreme past which allows it to converge rapidly after finding a convex minima [Goodfellow et al., 2016]. The RMSProp algorithm is presented in Table 2.5.

Table 2.5: RMSProp Optimisation algorithm [Goodfellow et al., 2016].

RMSProp Algorithm The RMSProp optimisation algorithm

Require: Global learning rate ϵ, decay rate τ
Require: Initialise Parameter $\boldsymbol{\theta}$
Require: Small constant δ to stabilise division by small numbers
start
  Initialise gradient accumulation variable $\mathbf{r} = 0$
  while stopping criteria not met **do**
    Sample a minibatch m examples from the training set $\mathbf{X}$, $\{\mathbf{X}_1, ..., \mathbf{X}_m\}$, with corresponding targets $\mathbf{Y}_i$
    Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{X}_i; \boldsymbol{\theta}), \mathbf{Y}_i)$
    Accumulate squared gradient: $\mathbf{r} \leftarrow \tau \mathbf{r} + (1 - \tau)\mathbf{g} \odot \mathbf{g}$
    Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
end

The RMSProp algorithm introduces a new hyperparameter τ that controls the length scale of the moving average. Empirically, RMSProp has shown that it is an effective and practical optimisation algorithm for deep neural networks and is routinely employed by deep learning practitioners [Shaziya, 2020].

Adam

Finally, Adam is another adaptive learning rate optimisation algorithm and is presented in Table 2.6. The name *Adam* is derived from the phrase *adaptive moments*. In the context of the earlier

algorithms, it can be interpreted as a variant on the combination of RMSProp and momentum with a few important distinctions. The first, is that the Adam algorithm incorporates momentum directly as an estimate of the first-order moment (with exponential weighting) of the gradient. The simplest way to include momentum to the RMSProp algorithm is to apply it directly to the rescaled gradients. The second distinction is that Adam includes bias corrections to the estimates of both the first order moments (momentum term) and the second-order moments to account for their initialisation at the origin,. RMSProp also incorporates an estimate of the second-order moment, however it does not include the correction factor. Thus, unlike in Adam, it is possible for the RMSProp second-order moment estimate to demonstrate large bias early in training. In comparison to other optimisation algorithms, Adam is regarded as being fairly robust to the choice of hyperparameters [Goodfellow et al., 2016].

Table 2.6: Adam Optimisation algorithm [Goodfellow et al., 2016].

Adam Algorithm The Adam optimisation algorithm

Require: Global learning rate ϵ
Require: Exponential decay rates for moment estimates, β_1 and β_2 in $[0, 1)$.
Require: Small constant δ to stabilise division by small numbers
Require: Initialise Parameter $\boldsymbol{\theta}$
start
  Initialise 1^{st} and 2^{nd} moment variables $\mathbf{s} = 0, \mathbf{r} = 0$
  Initialise time step $t = 0$
  while stopping criteria not met **do**
    Sample a minibatch m examples from the training set $\mathbf{X}$, $\{\mathbf{X}_1, ..., \mathbf{X}_m\}$, with corresponding targets $\mathbf{Y}_i$
    Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{X}_i; \boldsymbol{\theta}), \mathbf{Y}_i)$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\mathbf{s} \leftarrow \beta_1 \mathbf{s} + (1 - \mathbf{s})\mathbf{g}$
    Update biased second moment estimate: $\mathbf{r} \leftarrow \beta_2 \mathbf{r} + (1 - \beta_2)\mathbf{g} \odot \mathbf{g}$
    Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\beta_1^t}$
    Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\beta_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{+\delta}}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
end

The decay rates of the first and second order moment estimates are controlled by parameters β_1 and β_2. [Kingma and Ba, 2017] suggest that good default settings when using the Adam optimisation algorithm for most ML applications are $\epsilon = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and

$\delta = 10^{-8}$.

2.4.2 Regression and Classification loss functions

In the context of an optimisation algorithm, the function used to evaluate a candidate solution (i.e. a set of weights) that is being minimised or maximised is referred to as the objective function or criterion. Typically with neural networks, the primary goal is to minimise the error between the value predicted by the neural network and the desired value. The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number — a scalar value — which allows candidate solutions to be compared and ranked [Reed and Marks, 1998]. When calculating the error of the model during the optimisation process, a loss function must explicitly be chosen. This can be a challenging problem as the elected loss function must capture the properties that represent the design goals of the model.

Broadly speaking, loss functions can be classified into two major categories depending upon the type of learning task being considered - *Regression losses* and *Classification losses*. During classification tasks, a model is trained to predict the correct output from a set of finite categorical values - that is, for the MNIST handwritten digit dataset categorising the output into the correct integer class from 0 to 9. Regression, on the other hand, considers the prediction of a continuous quantity such as a price or magnitude.

Regression Loss Functions

Mean Squared Error (MSE) is perhaps the simplest and most common loss function used for regression tasks and is measured as the average of the squared difference between predictions and actual observations. The result is always positive regardless of the sign of the predicted and actual values due to the nature of the square and a perfect MSE returns a value of 0. The MSE is formally defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{y}_i)^2 \tag{2.13}$$

where n denotes the total number of training examples, i the current training sample from a training set input to the model, Y_i the target output for a given i and $\hat{y}_i$ represents the models prediction. A low MSE ensures that the output from the trained model contains no outlier predictions with large errors. To that end, if the model makes a single poor prediction, the error is magnified and a single poor prediction can have a large influence of the evaluated performance of the model due to the squaring nature of the function.

Classification loss functions

Cross-entropy is often considered the default loss function to use for classification problems and yields faster training and improved generalisation than sum-of-squares methods [Bishop, 2006]. It measures the difference between probability distributions for a given random variable or set of events. In information theory, the *surprise* of an event is often described. It is suggested that an event is more *surprising* the less likely it is and therefore contains more information. Similarly, an event is less surprising the more likely it is and subsequently contains less information [Wiley, 2005]. A binary classification problem defines a task of predicting one of two class labels for a given example i.e. assigning a correct label of 0 or 1 for a given input sample. To that end, a model can be used to estimate the probability that a sample belongs to a particular class label. Specifically, cross-entropy can be used to calculate the difference between two probability distributions. In binary classification tasks the target probability distribution P for an input returns an integer value of 0 or 1, where 0 refers to an impossibility of an event happening and 1 corresponds to a certainty that that event will happen. However, the integer labels contain no *surprise* at all and therefore, in the context of information theory, possess little information content (zero entropy). A model therefore seeks to approximate the target probability and its approximation is denoted by Q. In the language of classification, this refers to the actual probability Y and to the predicted probabilities $\hat{y}$, where Y refers to the known probability (target) of each class label for an example in the training set $\mathbf{X}$ and $\hat{y}$ denotes the probability of each class label for an example predicted by the model. The cross-entropy loss, L_{CE}, for a model is minimised across the entire training set $\mathbf{X}$ which is calculated by determining the average cross-entropy across all training examples such that

$$L_{\text{CE}}(Y, \hat{y}) = -\frac{1}{n}\sum_{i=0}^{n}\left[Y \log\left(\hat{y}_i\right) + (1 - Y)\log\left(1 - \hat{y}_i\right)\right] \tag{2.14}$$

where n is the total number of samples in the training set and i denotes the current sample. Alternatives to cross-entropy loss functions for binary classification tasks include the hinge and the hinge squared loss functions. The hinge loss function was primarily developed for use with Support Vector Machines (SVM) models [Cortes and Vapnik, 1995] and is intended for use with binary classification where the target values are in the set $\{-1, 1\}$. It encourages examples to have the correct sign and thus results in larger error when there is a difference in the sign between actual and predicted class values. For an intended output $Y = \pm 1$, the hinge loss L_H for a prediction $\hat{y}$ can be written as

$$L_H(Y, \hat{y}) = \sum_{i=1}^{n}\left[\max\left(0, 1 - Y \cdot \hat{y}\right)\right] \tag{2.15}$$

Similarly, a variation of the hinge loss function exists, known as the squared hinge loss function and is used for 'maximum margin' binary classification problems. The squared hinge loss function is denoted by L_{H2} As its name suggests, it is similar to the standard hinge function but

incorporates a square term with the intention of smoothing the surface of the error function to improve numerical performance, that is

$$L_{H2}(Y, \hat{y}) = \sum_{i=1}^{N} \left[\max(0, 1 - Y \cdot \hat{y})^2\right] \tag{2.16}$$

2.5 Evaluation of Optimisers and Loss Functions

2.5.1 Performance evaluation of optimisers

This section benchmarks the performances of each of the optimisers presented in Section 2.4 on the MNIST [Lecun et al., 1998] and CIFAR10 [Krizhevsky and Hinton, 2009] datasets. The MNIST dataset is a database of handwritten digits consisting of 60,000 28×28 black and white training examples and a test set containing 10,000 samples [Lecun et al., 1998]. Here, the test set is used to evaluate the performance of the optimisers. A MLP was trained for a total of 30 epochs to predict the correct class from $[0 - 9]$ when provided with an input image. The input image is flattened into a vector and is passed through a hidden layer containing 250 hidden nodes before entering a 10 node softmax layer for classification and the binary cross-entropy loss function was elected to train the model.

The CIFAR10 dataset consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images [Krizhevsky and Hinton, 2009]. The goal of the trained network is to correctly classify input images into one of 10 classes named *aeroplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* and *truck*. A Convolutional Neural Network (CNN) was trained on this dataset and consists of four 2D convolutional layers containing 32, 64, 16 and 32 filters respectively with convolutional layers 1 and 3 possessing a kernel size of 2×2 and layers 2 and 4 a kernel size of 3×3. The layers are flattened into a dense MLP layer containing 1,024 nodes before finally passing through a final MLP layer with 10 hidden nodes, which is activated by a softmax activation function for classification. Once again the binary cross-entropy loss function was used to evaluate the model and a CNN network was trained on each of the aforementioned optimisers. It should be noted that each optimiser is configured with the default hyperparameters in Keras and a learning rate of $\epsilon = 0.01$ was assigned for all cases. Momentum was not included for the analysis of the SGD optimiser and a δ value of e^{-7} was assigned for use with the AdaGrad, RMSProp and Adam optimisers. Finally, Adam parameters β_1 and β_2 were assigned values of 0.9 and 0.999 respectively.

Figure 2.10 plots the accuracy and loss of the optimisers during training. An accuracy score of 1 (100%) refers to a model that correctly classifies all samples in the test set (10,000 for

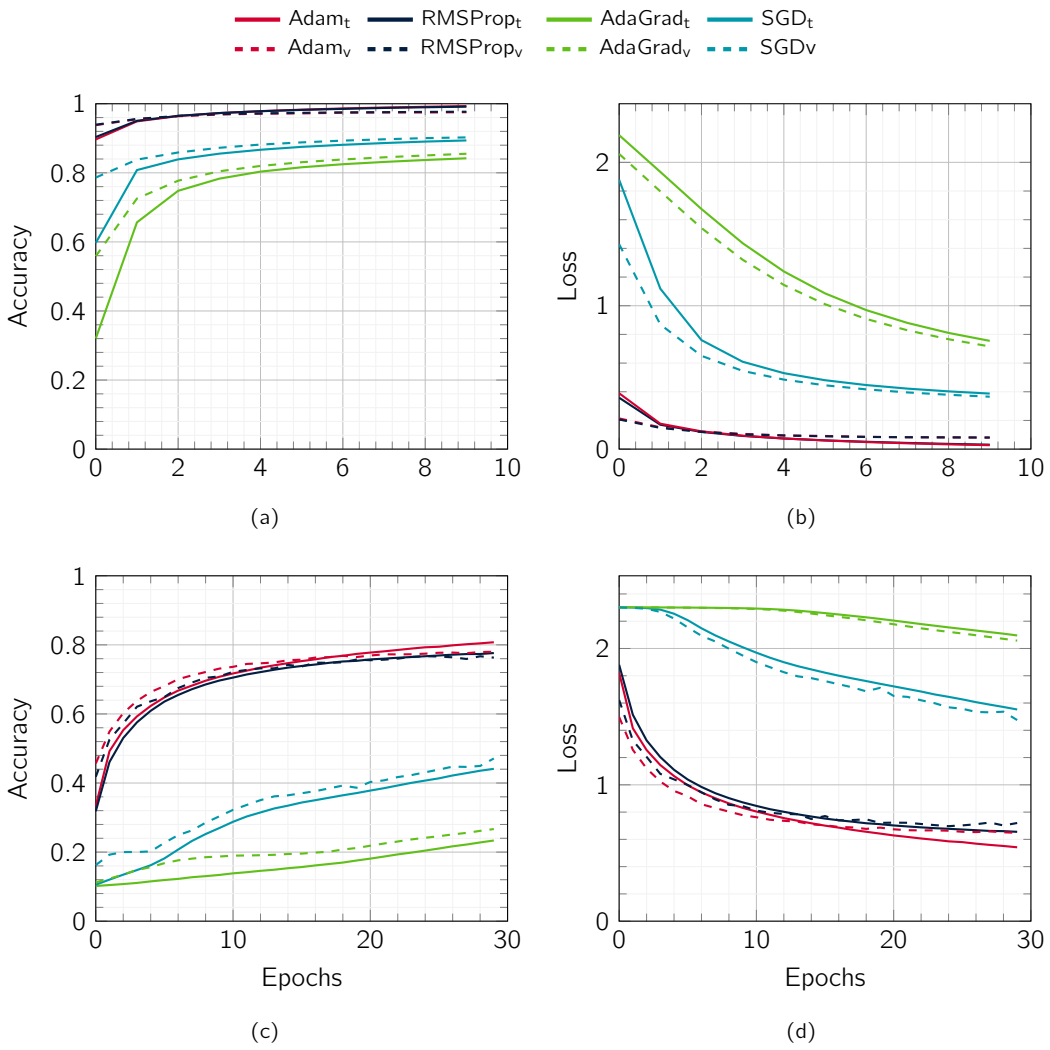


Figure 2.10: Performance comparison of Adam, RMSProp, AdaGrad and SGD optimisers where subscripts t and v denote the training and validation performance, respectively. Plots (a) and (b) show the accuracy and loss of the MLP model during training t and validation v on the MNIST dataset, respectively. Similarly, plots (c) and (d) show the accuracy and loss of the CNN model during training and validation on the CIFAR10 dataset, respectively.

each dataset) and the loss refers to the output from the loss function. Therefore a perfect model would return values of 1 and 0 for the accuracy and loss respectively. Despite MNIST often being regarded as a simple dataset to train neural networks on, there were some notable differences in performance by the optimisers. For the MNIST dataset, Figures 2.10(a) and 2.10(b), Adam and RMSProp were equivalent and the best performing optimisers both in terms of the accuracy and loss metrics. SGD was the worst performing model. Similarly,

in the larger CIFAR10 dataset the results follow the same trend. It can be seen that the performance of Adam and RMSProp are similar whereas once again SGD and AdaGrad are the worst performing models. It is worth noting that SGD might benefit from further training to improve performance as its accuracy has not yet converged. However, the discrepancy in performance between SGD compared to Adam/RMSProp is notable and it is very unlikely that the extra computational effort would yield more competitive results. Additional information regarding the final performance of the optimisers for the MNIST and CIFAR10 datasets are shown in Tables 2.7 and 2.8, respectively.

Table 2.7: MNIST final values after 10 epochs for SGD, AdaGrad, RMSProp and Adam optimisers.

Optimiser	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
Adam	99.56%	97.55%	0.021	0.081
RMSProp	99.44%	97.69%	0.023	0.081
SGD	89.54%	90.25%	0.380	0.366
AdaGrad	84.42%	85.52%	0.731	0.716

Table 2.8: CIFAR10 final values after 30 epochs for SGD, AdaGrad, RMSProp and Adam optimisers.

Optimiser	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
Adam	80.79%	78.01%	0.281	0.648
RMSProp	77.72%	76.31%	0.457	0.720
SGD	44.22%	47.11%	1.460	1.475
AdaGrad	24.56%	27.27%	2.054	2.058

2.5.2 Performance evaluation of loss functions for classification

This section considers the use of the aforementioned loss functions on a binary classification problem. Specifically, it investigates the circles test problem provided by scikit-learn [Pedregosa et al., 2011]. This problem considers samples drawn from two concentric circles on a 2D plane where points on the outer circle belong to a class assigned a label of 0 and points that belong to the inner circle are assigned a label of 1. Statistical noise in the region of 10% is introduced to the samples to add ambiguity to increase the difficulty of the problem and make the comparison more interesting. A total of $1,000$ samples were generated using scikit-learn's `make_circles` [Pedregosa et al., 2011] function of which 500 samples were used to train the model and the remainder used to validate its performance, as can be seen in Figure 2.11.

A MLP sequential model was defined using Keras' API that consisted of an input layer with two nodes for the respective x and y coordinate of the sample from the circle on a 2D plane.

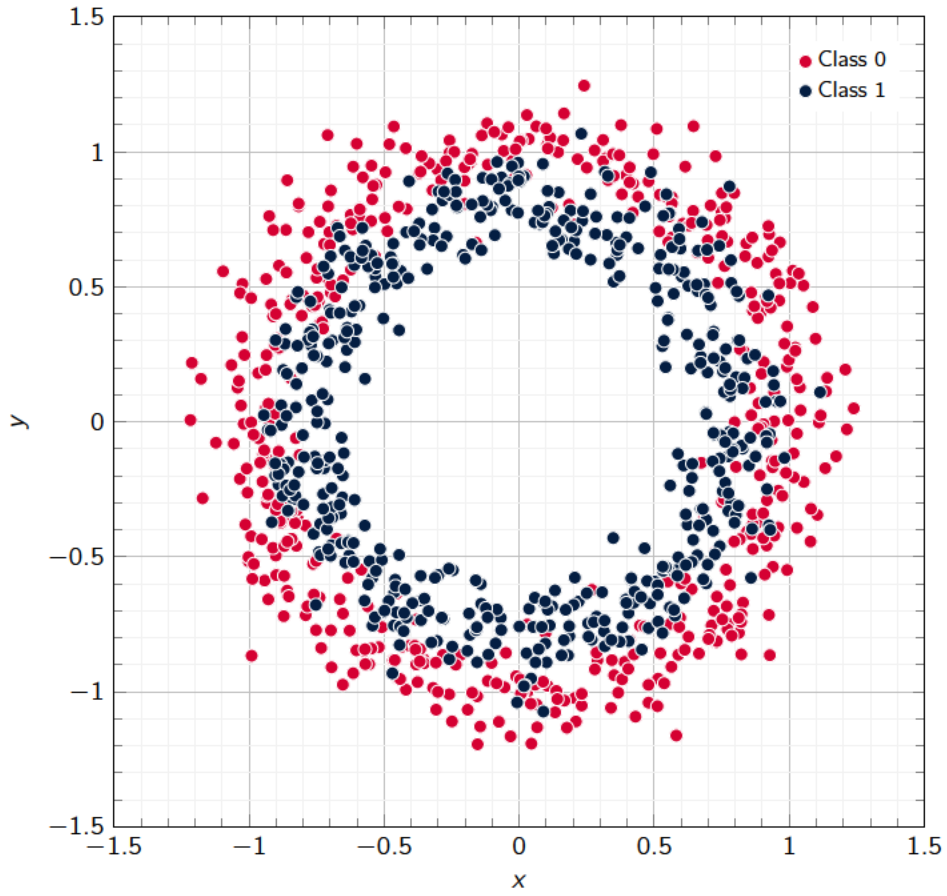


Figure 2.11: Training set used to compare the loss functions, consisting of 500 samples generated via scikit-learn's test problem that corresponding to concentric circles with class labels 0 and 1 [Pedregosa et al., 2011].

The hidden layer contains 40 nodes activated by a ReLU activation function and finally the output layer consists of a single node that corresponds to the classification prediction by the model which is activated by the sigmoid function to assure that the values lie between 0 and 1. A learning rate of 0.01 was selected and the model optimised using the SGD optimiser with momentum where $\gamma = 0.9$. A total of three models were trained to compare the performance when the model is compiled with a binary cross-entropy, hinge and hinge squared loss function. For the case of the hinge and hinge squared loss function, the class labels $\{0, 1\}$ were modified such that they belong to the set $\{-1, 1\}$ and the output layer activated by a tanh function instead to assure that its output lies between -1 and 1.

Figure 2.12 shows the performance of the models after 200 epochs of training on the training and validation set denoted by subscripts t and v, respectively, the results of which can be found in Table 2.9. It can be seen that in all cases the accuracy of the models improve

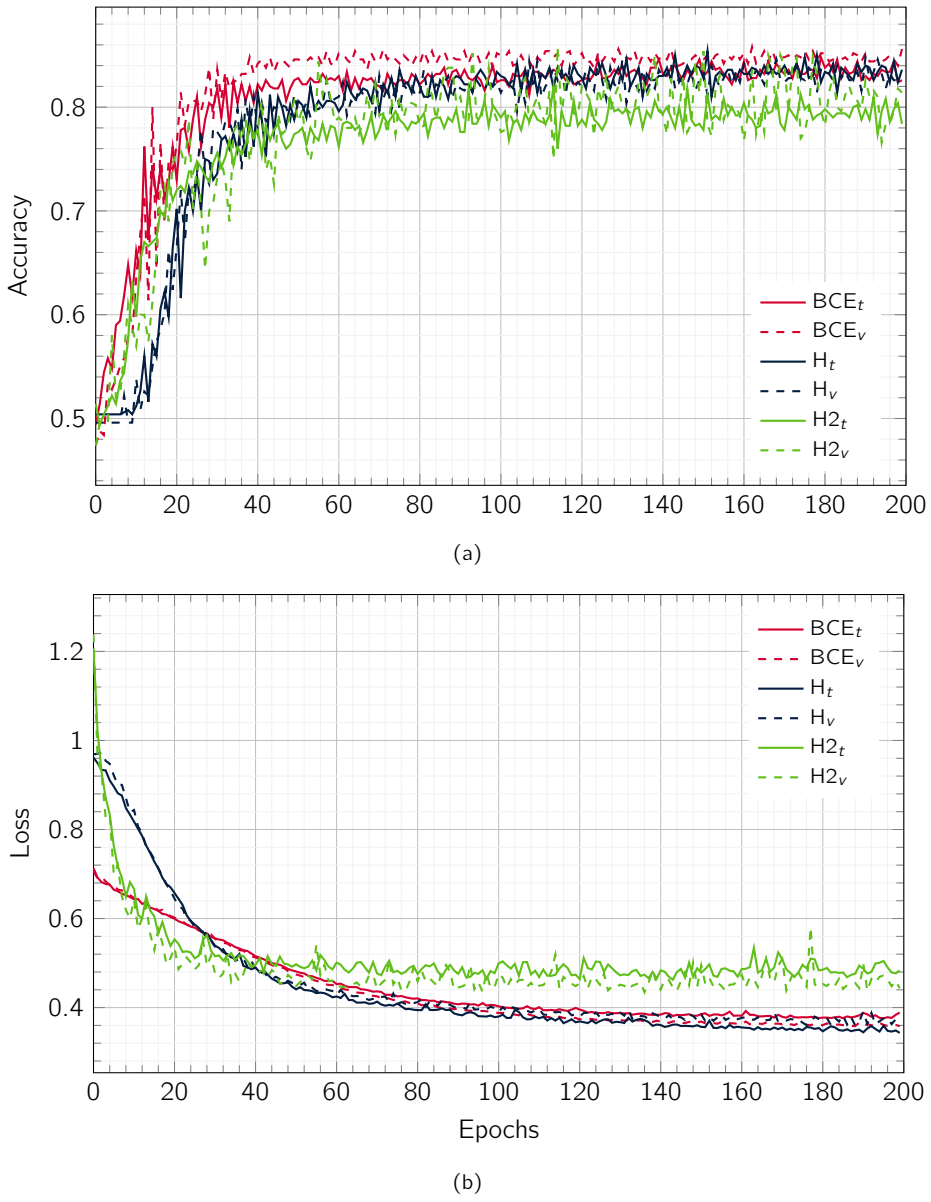


Figure 2.12: Comparison of MLP models compiled using the Binary Cross-Entropy (BCE), Hinge (H) and Hinge-Squared (H2) loss functions on scikit-learns circles problem [Pedregosa et al., 2011]. Where the respective performance on the training and validation set is denoted by subscripts t and v respectively.

with additional training and the binary cross-entropy loss function performed best converging at an accuracy of 84.2% and 85.6% for the training and validation set, respectively. When comparing the loss during training, it can be seen that the hinge-squared loss function converges fastest and after ≈ 60 epochs of training there are no obvious improvements in performance.

Conversely, the binary cross-entropy and hinge loss functions demonstrate similar performance and take longer to converge than the hinge-squared loss function, around ≈ 140 epochs, but converge to a lower loss. The reduced loss demonstrated by loss functions during training is echoed by its accuracy performance as both the binary cross-entropy and the standard hinge outperforms that of the hinge-squared loss function.

Table 2.9: Comparison of Binary Cross-entropy (BCE), Hinge (H) and Hinge-Squared (H^2) loss functions on scikit-learns circle problem test dataset [Pedregosa et al., 2011].

Loss Function	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
BCE	84.20%	85.60%	0.37	0.36
Hinge	83.20%	82.60%	0.34	0.36
$Hinge^2$	81.00%	81.40%	0.45	0.44

2.6 LSTM Networks

2.6.1 Overview

MLP networks have been presented in detail in the above sections. MLPs are the classical type of neural network that are well suited for use with tabular datasets and can be applied to many classification and regression prediction problems. A key attribute of the MLP network is its flexibility and they are generally used to learn the mapping between inputs and outputs. This flexibility allows them to be applied to other data types, for example pixels of an image can be flattened into a single vector of data and fed into the network. In practice, however, many types of neural network exist that have a different architecture that makes them better equipped to learn from different forms of data than a MLP network might be.

Convolutional Neural Networks (CNNs) for example, are a different class of neural network designed to map input data to an output variable. The benefit of using CNNs is their ability to develop an internal representation of a two dimensional image, which allows the model to learn position and scale in variant structures in the data. To that end, CNNs are regularly used for image classification prediction problems. More generally, CNNs are useful when a spatial relationship exists within the dataset. For example, an ordered relationship exists between words in a document of text and the time steps of a time series, and as such CNNs can also be applied to problems such as document classification and sentiment analysis [Yin et al., 2017a]. A Recurrent Neural Network (RNN) however, is a different class of neural network, specifically designed to work with time series or sequential data and is well equipped for sequence prediction problems. Sequence prediction problems come in many forms and are best described by the

types of inputs and outputs that they support. Some examples of sequence prediction problems include:

- One to many: An observation as input mapped to a sequence that contains multiple steps as an output.
- Many to one: A sequence of multiple steps as input mapped to a single prediction of class or quantity.
- Many to many: A sequence of multiple steps as input is mapped to a sequence that contains multiple steps as an output.

RNN algorithms are therefore well suited for ordinal or temporal problems such as language translation [Wu et al., 2016a], Natural Language Processing (NLP) [Yin et al., 2017b], speech recognition [Graves et al., 2013] and image captioning [Chu et al., 2020], and are incorporated into popular applications such as Siri, voice search and Google Translate [Wu et al., 2016b]. The architecture of a RNN is shown in Figure 2.13 and, similar to that of the MLP network, consists of an input layer, hidden layers and an output layer. A key difference between RNNs and traditional feed-forward networks is that adjacent nodes within the hidden layer of the RNN are connected and as such can share information at an intermediate level. Connections between the nodes of a RNN form a directed graph along a temporal sequence, which allows it to exhibit temporal dynamic behaviour. This trait is often termed *memory*, and RNNs are distinguished by their memory as they take information from prior inputs to influence the current input and output. This information is established as an internal state that can represent contextual information and, to that end, have the ability to retain information about past inputs for a duration that is not fixed a priori, but instead depends on its weights and input data. This is a notable difference, as traditional deep neural networks assume that inputs and outputs are independent of each other, whereas the output of RNNs depend on the prior elements in the sequence.

2.6.2 LSTM Layer Architecture

There are two widely known issues associated with the proper training of RNNs known as the vanishing and the exploding gradient problems [Pascanu et al., 2013]. All RNNs contain feedback loops in the recurrent layer that lets them maintain information in memory over time. In a network of n hidden layers, n derivatives will be multiplied together during training. If these derivatives are small then the gradient will decrease exponentially as it propagates through the model until it eventually vanishes — hence the vanishing gradient problem. Similarly, if the derivatives are large then the gradient will increase exponentially as it propagates through the model until it eventually *explodes* and is known as the exploding gradients problem. In the case

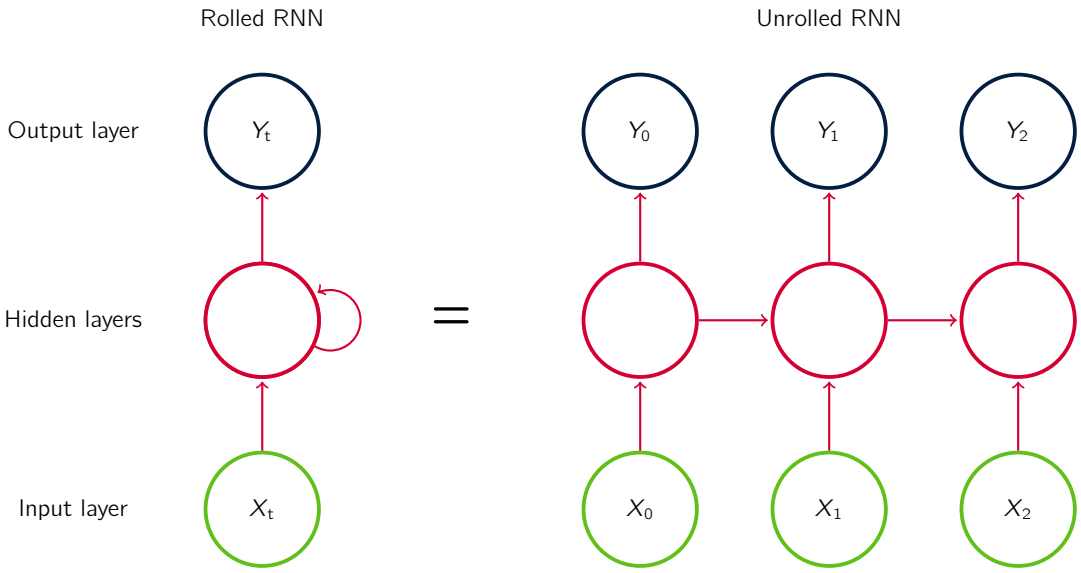


Figure 2.13: Unrolled RNN diagram.

of exploding gradients, the accumulation of large derivatives results in the model becoming unstable and incapable of effective learning. The large changes in the model weights creates an unstable network which at extreme values can cause the weights to become so large that it causes overflow, resulting in weight values that can no longer be updated. This causes *dead* neurons that can no longer contribute to the effective learning of the model. Conversely, the accumulation of small gradients results in a model that is incapable of learning meaningful insights given that the weights and biases of the initial layers will not be updated effectively. The worst case scenario is that the gradient falls to 0 at which point no further learning will take place. The Long Short Term Memory (LSTM) network is perhaps the most successful form of RNN as it overcomes some of the key training issues of traditional RNNs and they have been successfully applied on a wide range of applications [Sherstinsky, 2020].

LSTMs introduce the concept of cell states, which provide a pathway for the gradient to flow backward in time freely [Hochreiter and Schmidhuber, 1997a]. The main benefit of this is that it makes the network more resistant to the vanishing gradient problem.. The cell state acts as a transport highway that allows for information to be passed along different points of the network. In theory the cell state can transfer relevant information to any point during signal processing. This is important as it means that nodes that would be disconnected in traditional ANNs are now connected. This also means that information gathered from earlier time steps can be utilised at later time steps, reducing the effects of short term memory. As the cell state progresses through the network, it is modified and information may be added or removed from the cell state via gates. The gates are different neural networks that posses

parameters that are optimised during training to decide what information should be allowed on the cell state. Similar to traditional ANNs, the gates contain activation functions, specifically sigmoid activation functions. The sigmoid activation transforms the input vector it receives between values $[0, 1]$, which is useful when updating the cell state since information that returns a 0 value will be 'forgotten' (any number multiplied by 0 is 0). Similarly, any information that returns a value of 1 after passing through the sigmoid activation function is kept (any number multiplied by 1 is itself). To that end, information that returns a value closer to 1 is more influential and will have a greater effect on the output. For example, in sentiment analysis, it would be intuitive that strong adjectives such as 'amazing', 'beautiful' and 'incredible' would have a large influence on the cell state that progresses through the network, as they are strong indicators of positive sentiment.

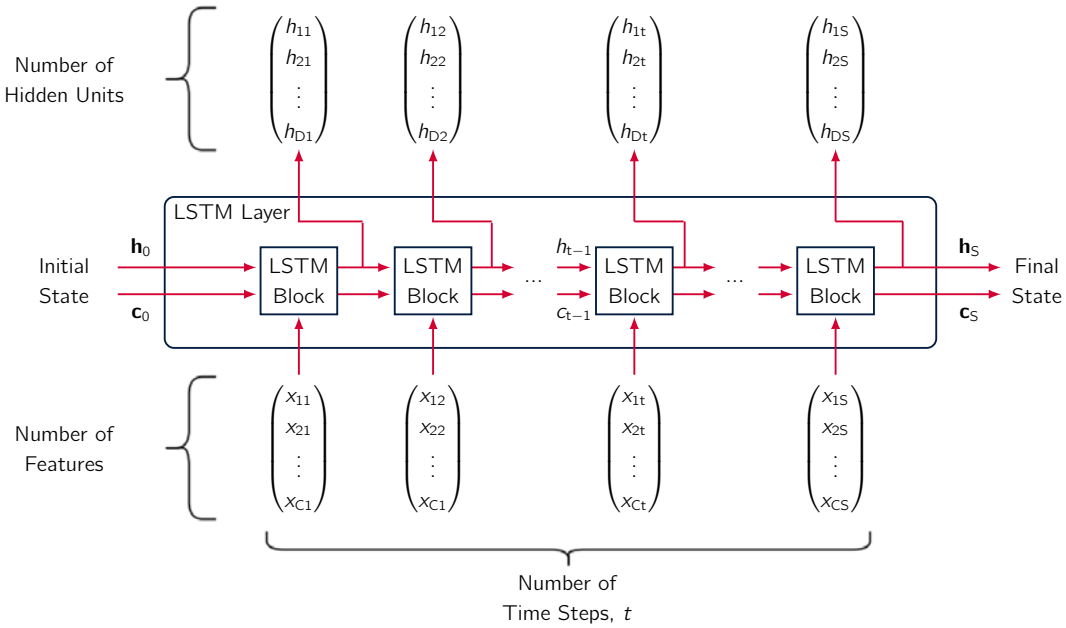


Figure 2.14: Diagram of the LSTM Layer Architecture indicating the flow of information through the entire LSTM network.

Figure 2.14 shows the architecture for a typical LSTM layer. It illustrates the flow of a time series X, with C features (channels) of length S through the layer. In the diagram, $\mathbf{h}_t$ and $\mathbf{c}_t$ denote the output (the hidden state) and the cell state at time step t, respectively. $\mathbf{h}_S$ and $\mathbf{c}_S$ refer to the final hidden state and cell state once the entire time series has passed through the network, respectively. The number of hidden units refers to the dimensionality of the hidden state and can be defined as a hyper-parameter when creating the LSTM architecture. The diagram illustrates how both the hidden state and cell state can pass through adjacent LSTM blocks/units in the network. Each LSTM block receives three inputs, the time series X and the hidden and cell states, and outputs updated versions of the hidden and cell states [Hochreiter

and Schmidhuber, 1997b].

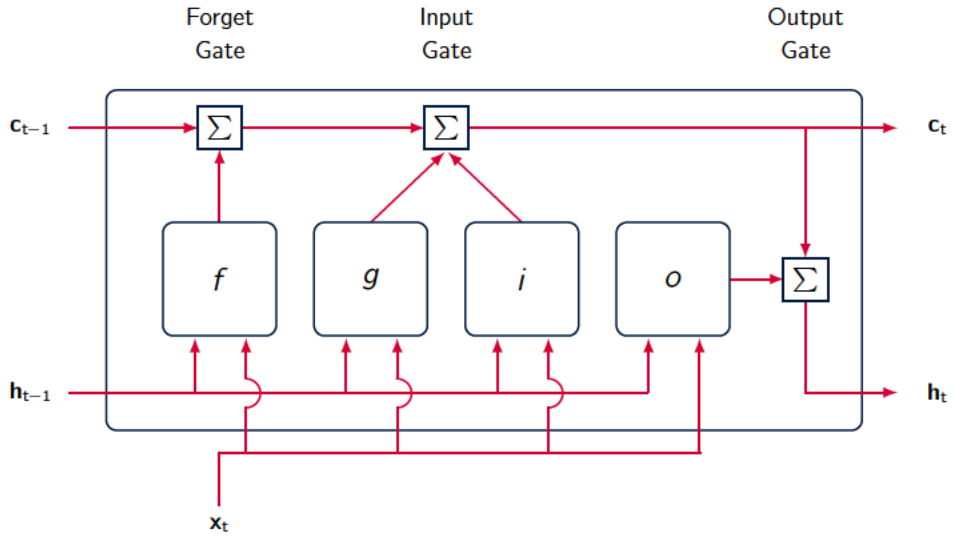


Figure 2.15: Diagram of a single LSTM unit.

Figure 2.15 illustrates the flow of data at time step t through a single LSTM unit. LSTMs use a series of gating mechanisms that controls the memorising process. Information in LSTMs can be stored, written, or read via gates that open and close. The learnable weights of an LSTM layer are the input weights $\mathbf{W}$, the recurrent weights $\mathbf{R}$ and the bias $\mathbf{b}$. The matrices $\mathbf{W}$, $\mathbf{R}$ and $\mathbf{b}$ are concatenations of the input weights, the recurrent weights, and the bias of each component respectively. The matrices are concatenated as follows:

$$\mathbf{W} = \begin{Bmatrix} W_\mathrm{i} \\ W_\mathrm{f} \\ W_\mathrm{g} \\ W_\mathrm{o} \end{Bmatrix}, \qquad \mathbf{R} = \begin{Bmatrix} R_\mathrm{i} \\ R_\mathrm{f} \\ R_\mathrm{g} \\ R_\mathrm{o} \end{Bmatrix}, \qquad \mathbf{b} = \begin{Bmatrix} b_\mathrm{i} \\ b_\mathrm{f} \\ b_\mathrm{g} \\ b_\mathrm{o} \end{Bmatrix}, \tag{2.17}$$

where subscripts i, f, g and o refer to the input gate, the forget gate, the cell candidate and output gate, respectively. The cell state at any time t is given by the following equation:

$$\mathbf{c}_t = F_t \odot \mathbf{c}_{t-1} + i_t \odot g_t \tag{2.18}$$

where $\odot$ refers to the Hadamard product (element-wise multiplication of vectors). The hidden state at time step t is given by:

$$\mathrm{h}_t = o_t \odot F_\mathrm{c}(\mathbf{c}_t) \tag{2.19}$$

where F_c is the state activation function. The transformations used to describe the components at time step t are listed in Table 2.10, where F_g denotes the gate activation function which most commonly uses the sigmoid activation function [Hochreiter and Schmidhuber, 1997b].

Table 2.10: Transformations applied at each gate within an LSTM cell.

Component	Formula
Input Gate	$i_t = F_g(W_i \mathbf{x}_t + R_i \mathbf{h}_{t-1} + b_i)$
Forget Gate	$f_t = F_g(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1} + b_f)$
Cell State	$c_t = F_c(W_g \mathbf{x}_t + R_g \mathbf{h}_{t-1} + b_g)$
Output Gate	$o_t = F_g(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1} + b_o)$

2.7 Predicting the response of a SDOF system

LSTM models are well suited to time-series problems and are a good candidate to be applied for engineering vibration problems. This example utilises an LSTM networks to predict the response of a linear single degree of freedom (SDOF) system . During the last decades, researchers have made increased efforts to propose effective structural health monitoring (SHM) techniques for civil buildings and structures [Nagamani Devi and Vijayalakshmi, 2021]. Estimation of the response of structures to vibration has been intensively investigated because it can provide useful information for inferring the health state of a structure as well as inherent structural characteristics. Using nonlinear time-history analysis, the performance level of a structure under various earthquake intensities can be determined via maximum drift estimation, i.e. by determining the maximum allowable lateral drift at the top of a building [Khouri, 2009]. For instrumented structures, data recorded from acceleration sensors can be used to compute the fundamental frequencies and mode shapes. Changes in the identified characteristics serve as an indicator for health assessment [Wu and Jahanshahi, 2019].

Recently, advances in ML techniques have led to additional research opportunities within the field of SHM. These ML approaches attempt to learn the underlying mechanisms from the available measurements and use that information to predict the possible outcomes given new input. Due to recent developments in computation hardware and sensor technology, the acquisition of data is much easier than it once was and thus increases the applicability and feasibility of ML approaches. This example considers the work of [Wu and Jahanshahi, 2019] who proposed a deep convolutional neural network (CNN) to estimate the dynamic response of a linear SDOF system and a non-linear SDOF system and references its performance against an MLP network. This example extends on this work by applying a LSTM network instead and compares the results with the work of [Wu and Jahanshahi, 2019].

In their research, [Wu and Jahanshahi, 2019] used vibration signals of the SDOF structure (i.e. the displacement, velocity, excitation, and acceleration time history) to train the neural networks. The networks were then used to evaluate a linear and a non-linear system adopted from [Masri et al., 2000], and considered two cases of input-output relationships: (1) the use

of acceleration, velocity and excitation to estimate displacement and (2) the use of excitation to estimate acceleration. In this example, the LSTM network will consider the second case and the model is trained such that it can predict the corresponding acceleration for any given excitation signal provided as input. Figure 2.16 shows an illustration of the SDOF system.

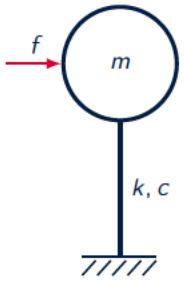


Figure 2.16: Illustration of the SDOF system.

2.7.1 Establishing the training set

The equation of motion of a linear SDOF system can be written as:

$$m\ddot{u} + c\dot{u} + ku = f \tag{2.20}$$

where $\ddot{u}$, $\dot{u}$ and u are the acceleration, velocity and displacement of the system, respectively, and m refers to the mass of the structure, k the stiffness, c is the damping and f the excitation of the system. The Newmark-beta integration method was used to solve the equation of motion and generate the training data used to train the networks. The Newmark-beta method is widely used in the numerical evaluation of structures and solids and was proposed by [Newmark, 1959]. By using the extended mean value theorem, the Newmark-beta method states that the first time derivative (velocity term in the equation of motion) can be solved as

$$\dot{u}_{n+1} = \dot{u}_n + \Delta t \ddot{u}_\Upsilon \tag{2.21}$$

where

$$\ddot{u}_\Upsilon = (1 - \Upsilon)\ddot{u}_n + \Upsilon\ddot{u}_{n+1}, \qquad 0 \leq \Upsilon \leq 1 \tag{2.22}$$

where the subscript n refers to the n-th time-step (Δt) and β and Υ are parameters that define the variation of acceleration over that time step. It is therefore possible to substitute equation 2.21 into 2.22 to obtain

$$\dot{u}_{n+1} = \dot{u}_n + (1 - \Upsilon)\Delta t\ddot{u}_n + \Upsilon\Delta t\ddot{u}_{n+1} \tag{2.23}$$

Because acceleration also varies with time however, the extended mean value theorem must also be extended to the second time derivative to obtain the correct displacement such that

$$u_{n+1} = u_n + \Delta t\dot{u}_n + \frac{1}{2}\Delta t^2\ddot{u}_\beta \tag{2.24}$$

where again,

$$\ddot{u}_\beta = (1-2\beta)\ddot{u}_n + 2\beta\ddot{u}_{n+1}, \qquad 0 \leq 2\beta \leq 1 \tag{2.25}$$

The discretised structural equations can then be written as

$$\dot{u}_{n+1} = \dot{u}_n + [(1-\Upsilon)\Delta t]\ddot{u}_n + \Upsilon\Delta t\ddot{u}_{n+1} \tag{2.26}$$

$$u_{n+1} = u_n + \Delta t u_n + [(0.5-\beta)\Delta t^2]u_n + [\beta\Delta t^2]\ddot{u}_{n+1} \tag{2.27}$$

To initialise the numerical method, it is necessary to define some initial conditions: the SDOF system has a mass $m = 1$ kg, a stiffness $k = 200$ N/m and a damping coefficient $c = 1.5$ kg/s. The excitation f of the system is defined as white noise with zero mean and a variance of 1. The sampling frequency and data length was set to 200 Hz for a duration of 8,300 s, respectively. This results in a total of 1,660,000 data points which forms the entire data sample used to both test and train the model. Parameters Υ and β were assigned values of 0.5 and 0.25, respectively, to assure average constant acceleration between time steps were $\Delta t = 1/200 = 0.005$ s. The initial displacement and velocity was set such that $u_0 = \dot{u}_0 = 0$. To that end, Figure 2.17 presents a 1 s extract of the generated original vibration signals used in this example.

The training data used consists of a (100000×1) input vector from the white noise sample and a corresponding (100000×1) target vector that refers to the acceleration of the SDOF system. Eight test sets were also prepared to evaluate the performance of the trained models. Each test set also consists of a (100000×1) excitation input vector and its expected (100000×1) acceleration output vector. Each test set input was afflicted with a varying level of noise to investigate the robustness of the networks and used noise values of $0, 1, 2, 3, 5, 10, 20$ and 30%. For the test cases it is important to note the following:

- The test sets are different to the training set and have not been used to train the model;
- Test sets are curated from the same sample and differ only by the additional noise added;
- Noise has not been added to the target data (output) and are therefore the same for each test case.

2.7.2 Setup of the LSTM Model

Figure 2.18 shows a flowchart of the layers used in the LSTM network.

The LSTM network used in this example consists of 200 hidden units and was trained for 250 epochs, where a single epoch refers to a complete pass of the training data through the network. A single iteration is completed when one sample is passed through the network. A

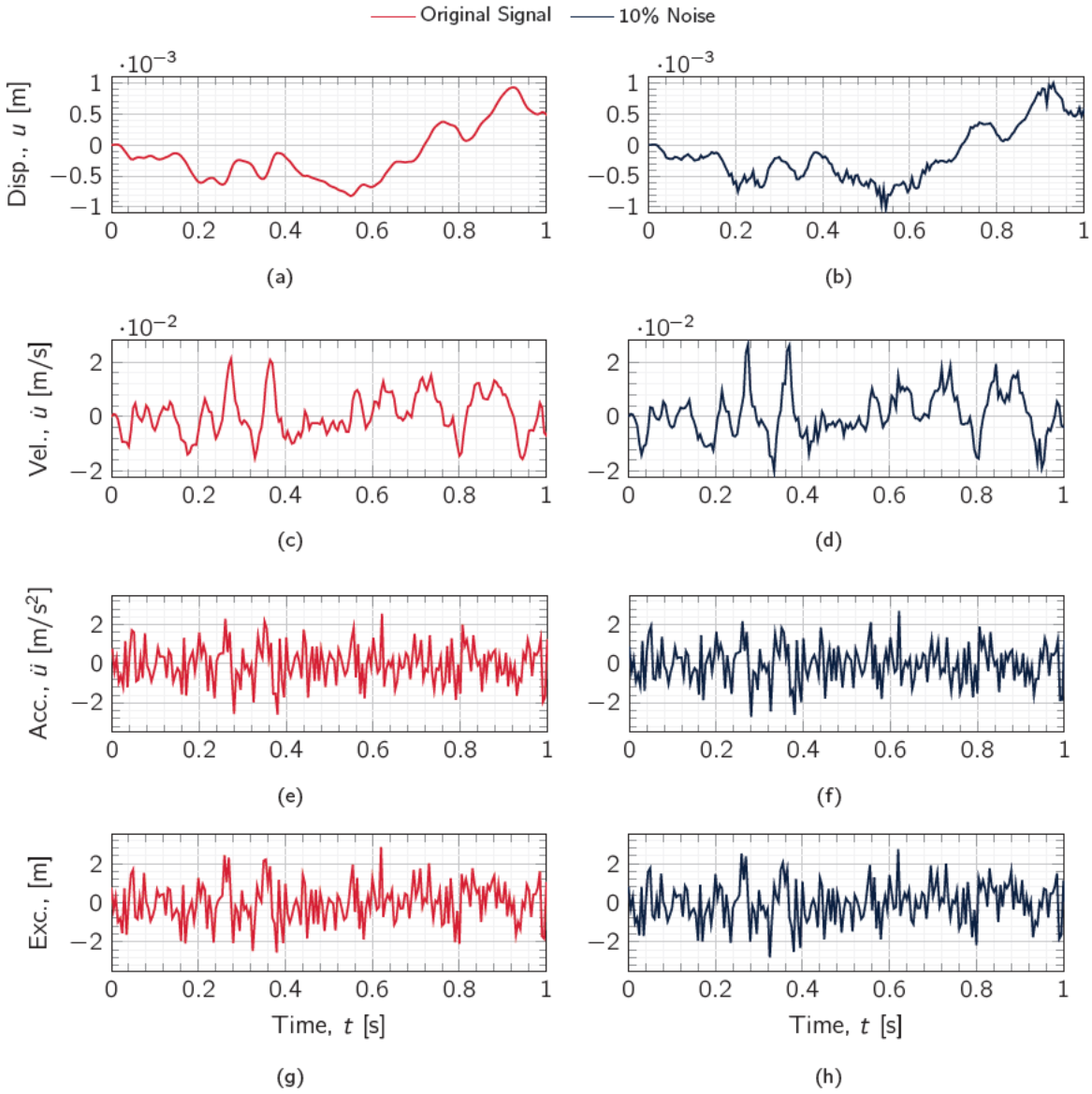


Figure 2.17: 1s sample Displacement, Velocity, Acceleration and Excitation of the linear SDOF system of the original signals and signals contaminated with 10% noise.

single sample consists of one feature (an excitation value) and one response (an acceleration value), and once each of the 100,000 samples in the training set pass through the network an epoch is complete. Each time a sample passes through the network, the weights are optimised via the Adam algorithm [Kingma and Ba, 2017]. The LSTM layer utilises the hyperbolic tangent (tanh) activation function and the initial learning rate was selected as 0.005 and adopted a

Figure 2.18: Flowchart of LSTM architecture used for regression analysis.

piecewise learning rate schedule which reduced the learning rate further by a factor of 0.125 every 125 epochs. This technique reduces the learning rate as training progresses and can often help the network learn finer details in the training set by reducing the vigor at which weights in the network are optimised. The initial values of the weights are initialised via the Glorot, also known as Xavier, initialiser [Glorot and Bengio, 2010] which samples weights from a uniform distribution with bounds $[-\sqrt{\frac{6}{N_0+N_i}}, \sqrt{\frac{6}{N_0+N_i}}]$ where N_i is equal to the number of hidden units (200) and $N_0 = 4N_i$. Since the LSTM model contains 200 hidden units, for a single sample its output is a (200×1) vector. A fully connected (FC) layer is therefore incorporated to convert the LSTM output into a single output i.e. prediction.

2.7.3 Results

Figure 2.19 shows a 0.5 s sample acceleration output from the trained LSTM network for test cases with 0, 10, 20 and 30% added noise and compares them with the expected true value. It can be seen that the LSTM prediction successfully captures the wave shape of the ideal target for each of the noise cases. It also appears that the output from the LSTM model demonstrates a slight bias to under-predict the acceleration of the SDOF system. Although studying the LSTM output visually is a useful tool to interpret the success of the model, objectively the 0.5 s sample is not sufficient to make any substantial conclusions about the performance of the network. To that end, Figure 2.20 depicts the error distribution of the LSTM for noise cases 0, 10, 20 and 30% and are shown in Figures 2.20 (a) to (d), respectively. A fitted normal distribution is superimposed on the histogram for comparison. The residuals of the 0% noise case lie within a tall bell-shaped curve that does not demonstrate any signs of skew. Its residuals lie between -0.082 and 0.055 with a standard deviation of 0.013. The distribution peaks at a density of ≈ 30, which is higher than that of the remaining noise cases and the distribution is also much narrower. Given that the count of residuals is consistent for each noise case, this indicates that a larger proportion of the predictions for the 0% noise case are closer to the expected value and is therefore more accurate.

As expected, the error distributions increase in width as the input signal is afflicted with more noise and it can be seen that the standard deviations across the noise samples increase from 0.013 to 0.086 for 0 and 30%, respectively. The distribution plots for each noise case follow a normal distribution with no signs of skew. Table 2.11 presents relevant statistical

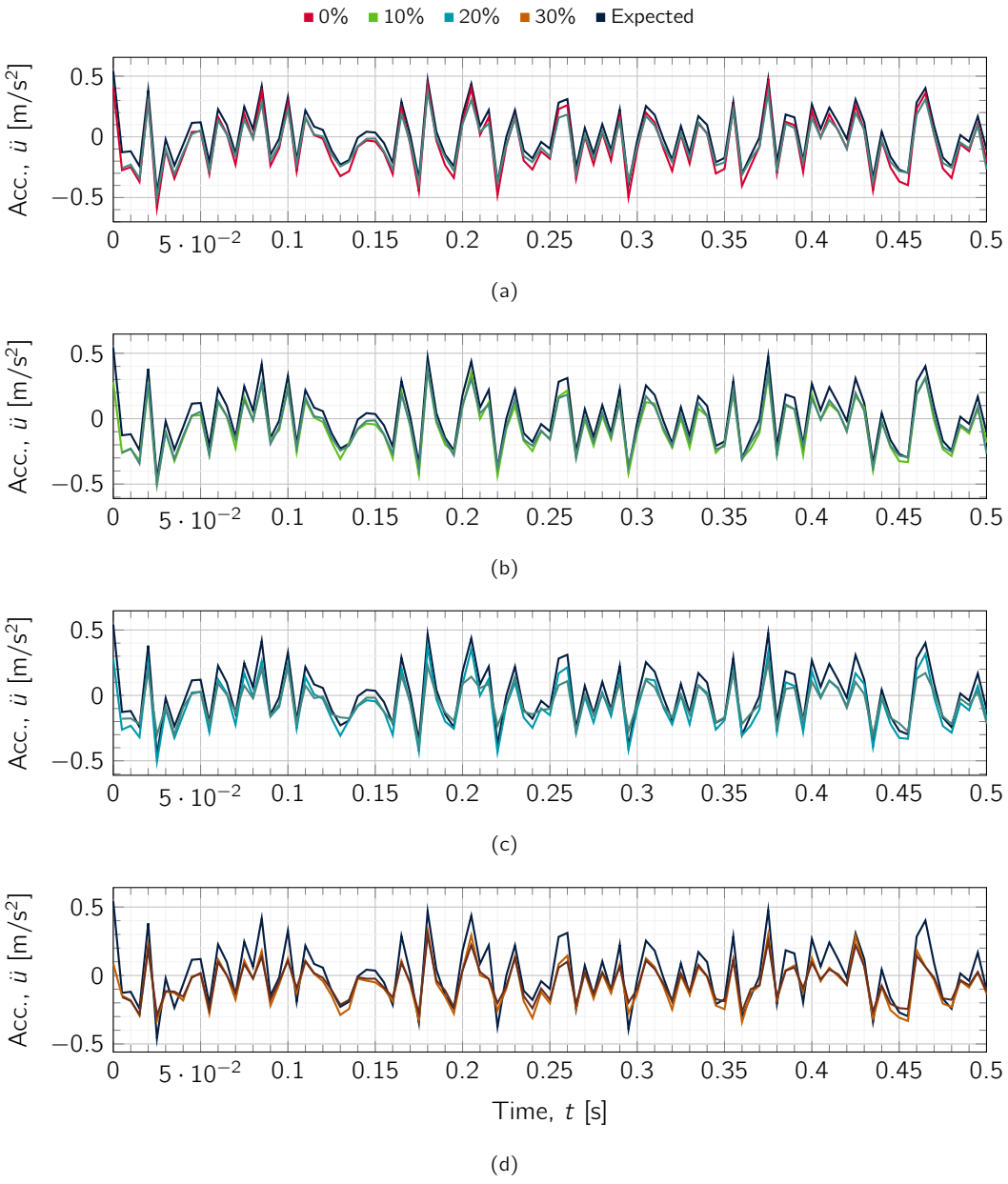


Figure 2.19: Predictions by the trained LSTM model for the 0%, 10%, 20% and 30% noise cases compared with the expected output.

information regarding the performance of the model for each input case. It can be seen that the model has a tendency to under predict the true acceleration as the mean value, $\bar{x} = -0.014$ for noise cases 0 to 20% and $\bar{x} = -0.013$ for 30%. On inspection of Figure 2.19, it can be

Table 2.11: Residual analysis of output from the trained LSTM network.

Noise [%]	$\bar{x}$	$std(x)$	x_{min}	x_{max}	RMSE
0	−0.014	0.013	−0.082	0.055	0.019
1	−0.014	0.012	−0.089	0.051	0.019
2	−0.014	0.016	−0.110	0.079	0.021
3	−0.014	0.010	−0.106	0.071	0.017
5	−0.014	0.023	−0.191	0.131	0.027
10	−0.014	0.041	−0.271	0.231	0.044
20	−0.014	0.072	−0.533	0.501	0.074
30	−0.013	0.086	−0.739	0.716	0.087

seen that even for higher noise cases the predicted accelerations follow the wave shape of the ideal target acceleration. In addition, the RMSE for the 30% noise case returned the largest value of 0.087, which equates to a relative difference of $\approx 8.7\%$. Figure 2.21 compares the RMSE predictions from the LSTM network with that of the MLP and CNN networks applied by [Wu and Jahanshahi, 2019]. It can be seen that the trained LSTM model returns lower RMSE values than both the MLP and CNN network for each of the noise cases.

2.7.4 Conclusion

This short example presented a LSTM-based approach for the vibration response estimation of a linear SDOF system. The study was inspired by the work of [Wu and Jahanshahi, 2019] who performed a similar analysis using MLP and CNN networks, and considered a test case where the excitation applied to a SDOF was used to predict the subsequent acceleration of the system. The excitation was defined as white noise with zero mean and a variance of 1 and the input samples were afflicted with additional noise in the range $0 - 30\%$ to account for measurement in real-world conditions and used to test the stability of the network. It was found that the LSTM network is well suited for use with engineering time-series data as the trained model was successfully able to predict the acceleration response of the SDOF system to an RMSE error of $1.9 - 8.7\%$ across the noise range of $0 - 30\%$. This was an improvement on the predictions made by the MLP and CNN networks presented by [Wu and Jahanshahi, 2019] and a testament to the potential that LSTM models can bring within the field of SHM. The LSTM model demonstrated that it is robust and capable of making accurate predictions against noise-contaminated data. The final part of this chapter introduces GANs which are the principle model used in this thesis.

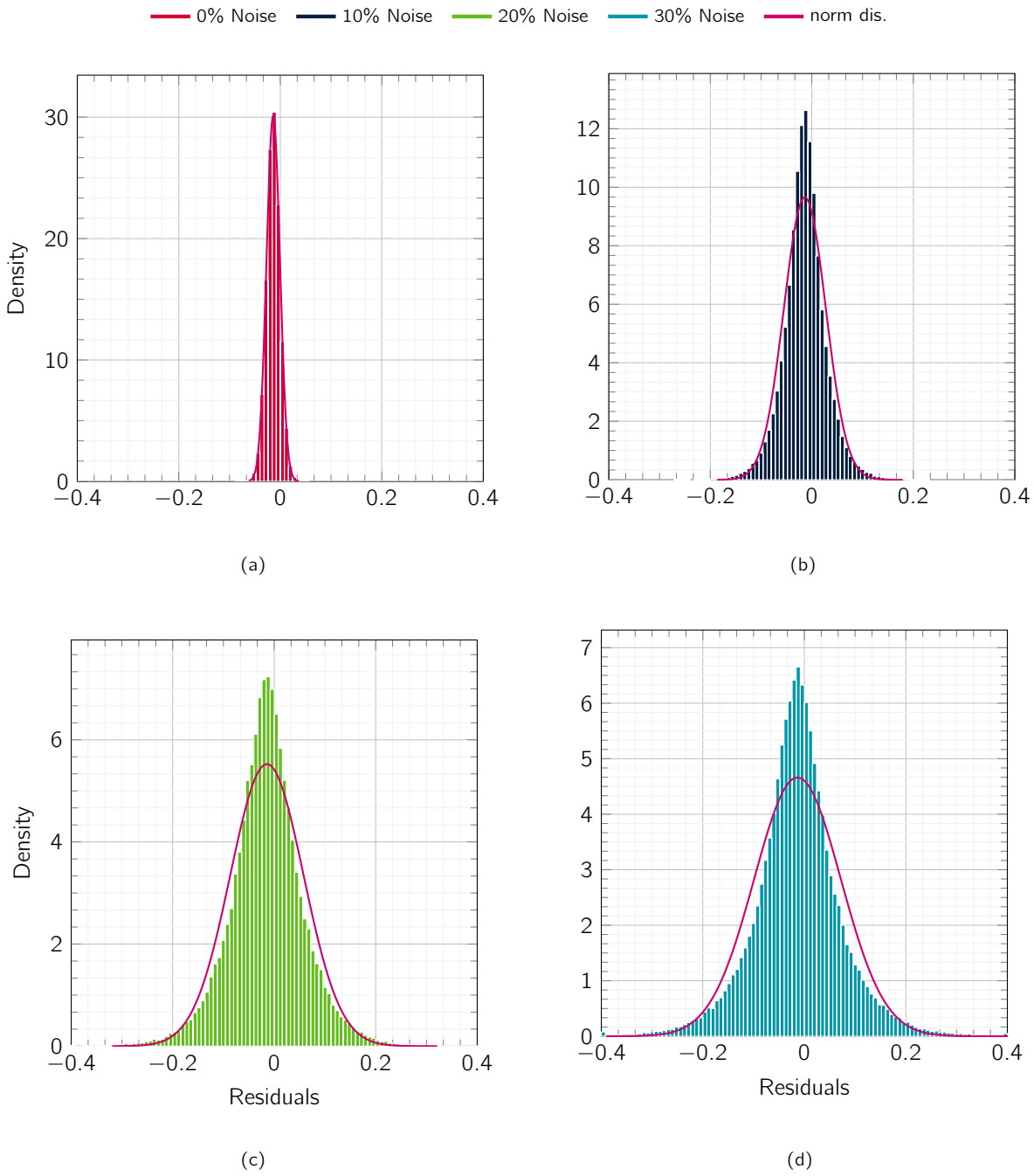


Figure 2.20: Density histogram of LSTM error residuals for noise cases 0%, 10%, 20% and 30%. The respective normal distribution curve is also plotted for each noise case.

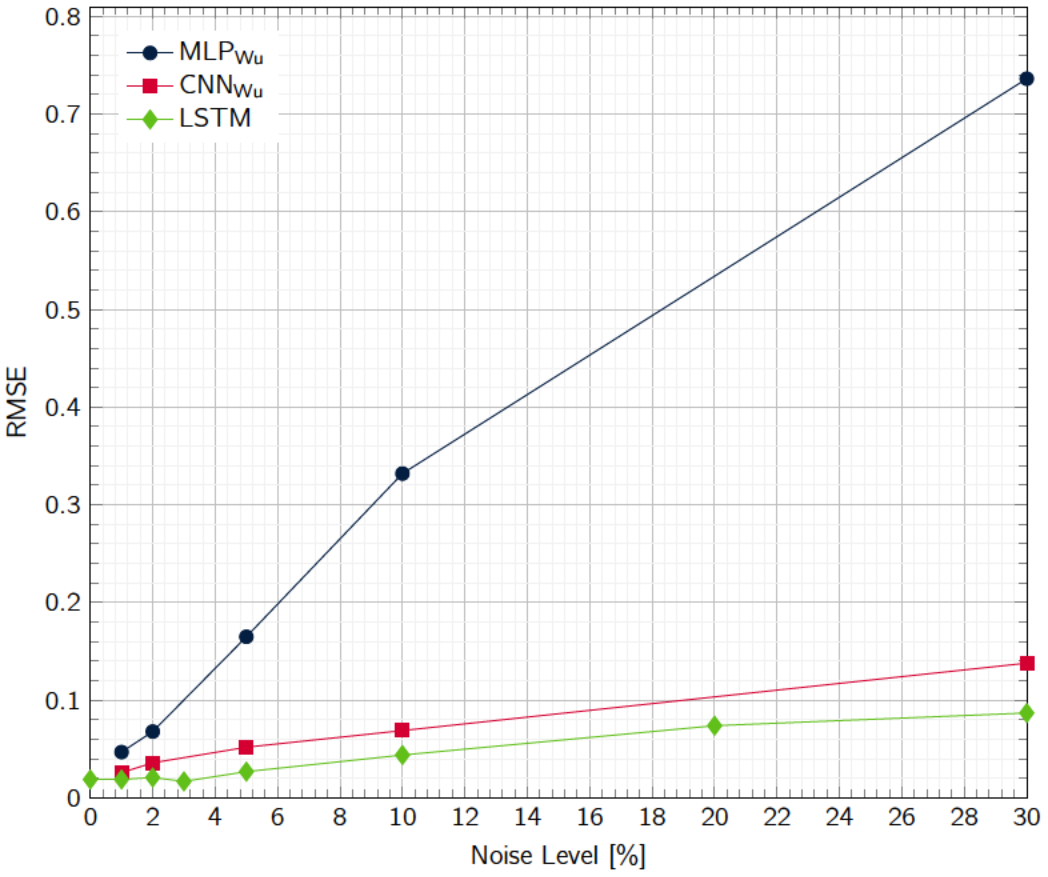


Figure 2.21: RMSE comparison for SDOF system with MLP/CNN networks from [Wu and Jahanshahi, 2019] and the LSTM network.

2.8 Generative Adversarial Networks

Finally, this section introduces the primary model of interest in this thesis which is the GAN. GANs are an approach to generative modelling using deep learning methods first proposed by [Goodfellow et al., 2014]. Generative modelling is an unsupervised learning task in machine learning that challenges the model to learn patterns and regularities in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset [Salimans et al., 2016]. Unsupervised learning exhibits self-organisation that captures patterns as neuronal predilections or probability densities [Hinton and Sejnowski, 1999]. This is in contrast to the models that have been discussed earlier which learn via supervised learning in which the training data is labelled in advance and maps an input to an output based on example input-output pairs. GANs are a clever way of

training a generative model by framing the problem as a supervised learning problem with two sub-models: a generator, G, and a discriminator, D. The discriminator D is a classification network optimised to assign the correct labels to real (label $= 1$) samples, i.e. those that came from the training set and fake samples (label $= 0$). Meanwhile, the generator G is trained to generate fake samples that fool D into believing that they came from the original training set [Chen et al., 2015]. In other words, D and G play the following two-player minmax game with value function $V(G, D)$ [Goodfellow et al., 2014]

$$\min_G \max_D V(D, G) = E_{\mathsf{X}} \log D(\mathbf{X}) + E_{\mathsf{Z}} \log[1 - D(G(\mathbf{Z}))] \tag{2.28}$$

where $\mathbf{X}$ is the input to D from the training set, $\mathbf{Z}$ is a vector of latent values input to G, E_{X} is the expected value over all real data instances, $D(\mathbf{X})$ is the discriminator's estimate of the probability that real data instance from $\mathbf{X}$ is real, E_{Z} is the expected value over all random inputs to the generator and $D(G(\mathbf{Z}))$ is the discriminator's estimate of the probability that a fake instance is real. The primary goal of G is to fool D and produce samples that D believes come from the training set. The primary goal of D is to assign a label of 0 to generated samples, indicating a fake, and a label of 1 to true samples, that is, samples that came from the training set. The training procedure for G is to maximise the probability of D making a mistake, i.e. an incorrect classification. In the space of arbitrary functions G and D, a unique solution exists, with G able to reproduce data with the same distribution as the training set and the output from $D \approx 0.5$ for all samples, ultimately indicating that the discriminator can no longer differentiate between the training data and data generated by G.

A GAN is therefore able to estimate generative models via an adversarial process, in which two neural networks compete against one another and are trained together. Given a training set, the G model learns to generate entirely new data with the same statistical representation as the training set.

The most common applications of GANs are used in the domain of image generation and editing. Figure 2.22 shows an example of photo-realistic images generated by [Karras et al., 2018a] using a GAN model that contains CNN layers trained on the CelebA-HQ dataset [Karras, 2017]. The CelebA-HQ dataset is a high quality version of the traditional CelebA dataset and contains $30,000$ high quality images of celebrities at a resolution of 1024×1024. This work demonstrates the capabilities of the GAN algorithm to learn characteristic features from a training set and generate convincing new samples that could plausibly belong to that original training set. GANs have also been used to generate cartoon characters [Jin et al., 2017], perform image-to-image translation [Isola et al., 2018] and text-to-image translation [Zhang et al., 2017]. [Ledig et al., 2017] developed their Super-Resolution GAN (SRGAN) model to generate output images with increased pixel resolution when compared to the input. Whilst the vast majority of GAN applications include some form of CNN layers to facilitate work with image data, it is the specific architecture of the GAN that determines its functionality and compatibility with different data types. An interesting variant on the traditional GAN is the

conditional-GAN (cGAN) first developed by [Mirza and Osindero, 2014] which allows a standard GAN network to be conditioned on auxiliary information during training — this is particularly useful as it addresses a common issue with GANs where there is limited control over the output.

Figure 2.22: An example of work conducted by [Karras et al., 2018a] that produces state-of-the-art photo-realistic human images of size 1024x1024. All images were generated by StyleGAN [Karras et al., 2018b] trained on the CelebA-HQ dataset.

2.8.1 Training a GAN

Figure 2.23 presents a flowchart with the training procedure of a GAN, where the generator, G, takes a latent input, $\mathbf{Z}$. In this instance, a latent variable is a hidden or unobserved variable, and the latent space is a multi-dimensional vector of these variables. Often, the latent input to a G network might exist as a vector of Gaussian random numbers. the length of which is defined prior to training. The stochastic nature of the latent input is important as it means that variability is introduced to the G predictions. If the latent input is constant, then so is the output from G and the same prediction will be made. As G receives the latent input $\mathbf{Z}$, it makes a prediction $G(\mathbf{Z})$. The nature of $G(\mathbf{Z})$ depends on the architecture of the network and the training data, but new predictions are made by simply passing Z into the generator network. However, without any training $G(\mathbf{Z})$ is unlikely to be of any use. Similarly, it can be seen that D takes input from either the training data $\mathbf{X}$ or from $G(\mathbf{Z})$. D is a classification network trained to assign the correct label to real and fake samples. During training it receives examples of real samples from the training data $\mathbf{X}$, and fake samples from G in the form $G(\mathbf{Z})$. D therefore has two potential outputs, either $D(\mathbf{X})$ or $D(G(\mathbf{Z}))$ depending on the input it receives.

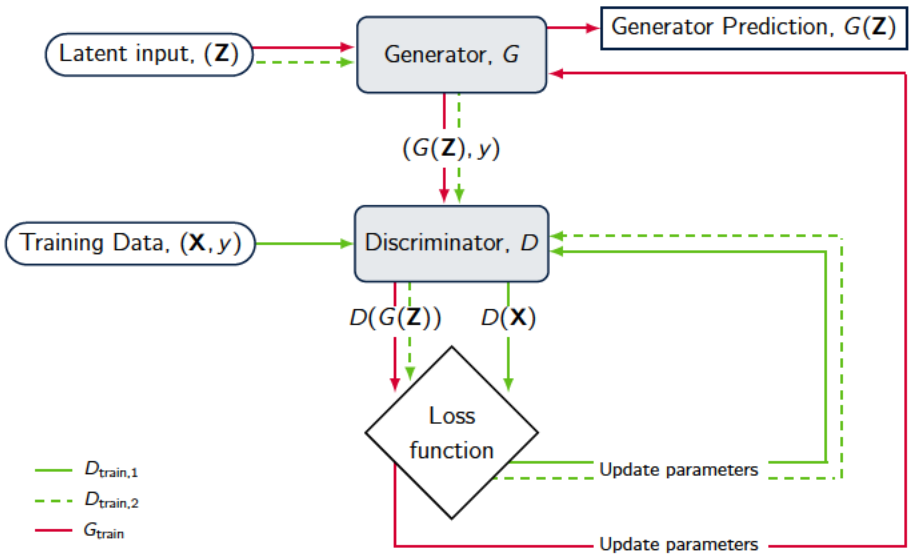


Figure 2.23: Flowchart of training processes for both the discriminator D and generator G within the GAN. Where $D_{\text{train},1}$ refers to the process of training the discriminator on real data from the training set (label = 0) and $D_{\text{train},2}$ refers to training on samples generated by G. G_{train} refers to the process of training G based on the output of D.

Training the Discriminator

A single iteration of training is complete once the trainable parameters of both the G and D networks have been updated on a total of n samples. The steps required to train the D network within one iteration is presented first. $n/2$ samples from the training set $\mathbf{X}$ are prepared and passed into D to obtain $D(\mathbf{X})$, this is denoted by the path $D_{\text{train},1}$ on Figure 2.23. A target label of 1 is concatenated with the training data to identify them as real samples. The output $D(\mathbf{X})$, i.e. D's prediction as to whether the sample is real or fake, then passes into a loss function which is then utilised by an optimiser to update the trainable parameters of D with the intention of changing D's output such that it returns a value of 0 for real samples. Similarly, D is then trained on $n/2$ fake samples generated by G and denoted in the flowchart by $D_{\text{train},2}$. As the D network is trained twice, the number of fake and real samples it is trained on is halved such that its total exposure is consistent with that of G at n total samples. A latent input $\mathbf{Z}$ is prepared and G is used to generate $n/2$ samples in the form $G(\mathbf{Z})$. $G(\mathbf{Z})$ is passed into D and the input is concatenated with a target label of 0 to denote the fake sample. $D(G(\mathbf{Z}))$ then passes through the loss function and the D network is optimised with the intention of returning a value of 1 for future fake samples.

Training the Generator

Once D has been updated on real and fake samples, the G network is trained. The training path is denoted by G_{train} on Figure 2.23. The goal of G is to generate samples that D believes came from the training set $\mathbf{X}$, therefore the output $D(G(\mathbf{Z}))$ is used to train and update the parameters of G. First, the latent input $\mathbf{Z}$ is generated and its predictions $G(\mathbf{Z})$ are passed into D. This time, however, it is concatenated with a label of 1 such that D believes that they are real samples. The output $D(G(\mathbf{Z}))$ is passed through the loss function and used to update the G network with the intention of $D(G(\mathbf{Z}))$ returning a value of 1 for fake samples. It is important to note that the D model is frozen during G_{train} and its parameters are not updated. Over time, the ability of G to generate convincing samples improves and the D network finds it more difficult to differentiate between real and fake samples.

2.8.2 Example: Training a GAN network on a sine wave

This section demonstrates the ability of a GAN model to learn features from a simple training set and generate new samples that belong to the same representation. Here, the training set $\mathbf{X}$ used corresponds to a sine wave, $y = sin(x)$, that consists of 50 samples where $0 \leq x \leq 2\pi$. Each sample exists as a 2 element vector and contains an x and y coordinate. The training set X is shown in Figure 2.24.

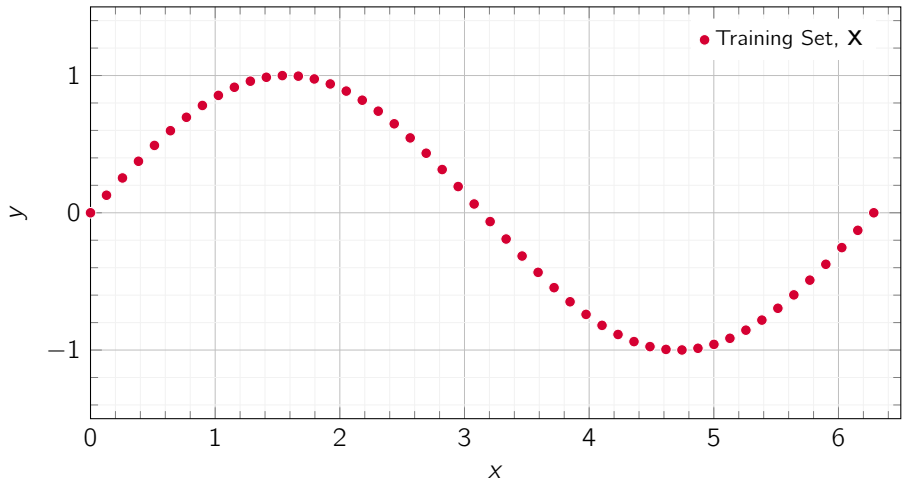


Figure 2.24: Training set $\mathbf{X}$ used to train the GAN model that consists of 50 samples. Each sample contains an x coordinate and its corresponding y coordinate.

The machine learning algorithm for the GAN was written in Python3, using TensorFlow's high-level Keras API for building and training deep learning models, and was used to define the

G and D networks. G is a MLP network that takes a latent vector $\mathbf{Z}$ of length 10 as input before passing through a hidden layer consisting of 15 hidden nodes. The output from the hidden layer passes through a LReLU activation function, where parameter α was set to a value of 0.01, before entering the output layer that consists of 2 hidden nodes, where each node refers to the x and y coordinate of G's prediction. The weights of the network were initialised via the He uniform initialiser [He et al., 2015b]. Since the G network is trained from the output of D, both networks are combined to form the GAN model and is compiled using the binary cross entropy loss function and optimised via the adam optimisation algorithm [Kingma and Ba, 2014]. The D network was also defined as a MLP network and takes 2 inputs that correspond to the x and y coordinate of either real and fake samples and outputs a single value that denotes its classification between 0 and 1 as to whether it believes the sample is real or fake. The network consists of 2 hidden layers containing 15 and 10 nodes, respectively, both of which were activated by LReLU activation functions where α was also set to a value of 0.01. The model was trained for a total of $50,000$ iterations and the output from the G network at different intervals during training is shown in Figure 2.25.

The G network was used to generate 500 samples at interval iterations of 500, $5,000$, $25,000$ and $50,000$ during the training process. Figure 2.25 includes the training set $\mathbf{X}$ used to train the GAN model and it can be seen that after 500 iterations the samples show no correlation. The samples generated are dispersed with no clear shape and lie within a domain of $-0.5 \leq x \leq 3$ and $-2 \leq y \leq 2$, which is inconsistent with that of $\mathbf{X}$. It is important to note that the quality of the samples generated by G are dependent on the classification accuracy of D. If D is unable to correctly classify samples that are real and belong to the training set, then nor will it be possible for G to generate samples that could have come from the training set. This is a key reason as to why the samples generated at early iterations, such as G_{500}, show little resemblance to that of the target. At $5,000$ iterations, it can be seen that the output from G appears more organised. Now the samples are less dispersed and fall within a domain of $1 \leq x \leq 6.5$ and $-1.3 \leq y \leq 1.5$, which is closer to that of $\mathbf{X}$. The generated samples appear to be around the central diagonal that connects the peak and trough of the sine wave, however it demonstrates no clear evidence of curvature which is a key feature of the sin wave. At $25,000$ iterations however, there is evidence that the performance of the model has improved with training. The samples generated at this iteration are less dispersed than at $5,000$ iterations and begin to display some of the key properties of the sine wave such as the stronger curvature at the peaks and troughs. Between $0 \leq x \leq 2$, the samples generated display strong correlation with the training set $\mathbf{X}$. For values of $x > 2$, the samples follow the general trend of the sin curve but with increased dispersion, particularly around $x = 4.5$. Finally, after $50,000$ iterations the output from G shows strong correlation with X across its entire domain. The trained G model is able to successfully generate samples across the entire domain of X and demonstrated its ability to learn key features of the sine waves such as the curvature of the peaks and troughs at x values of 1.57 and 4.71, respectively.

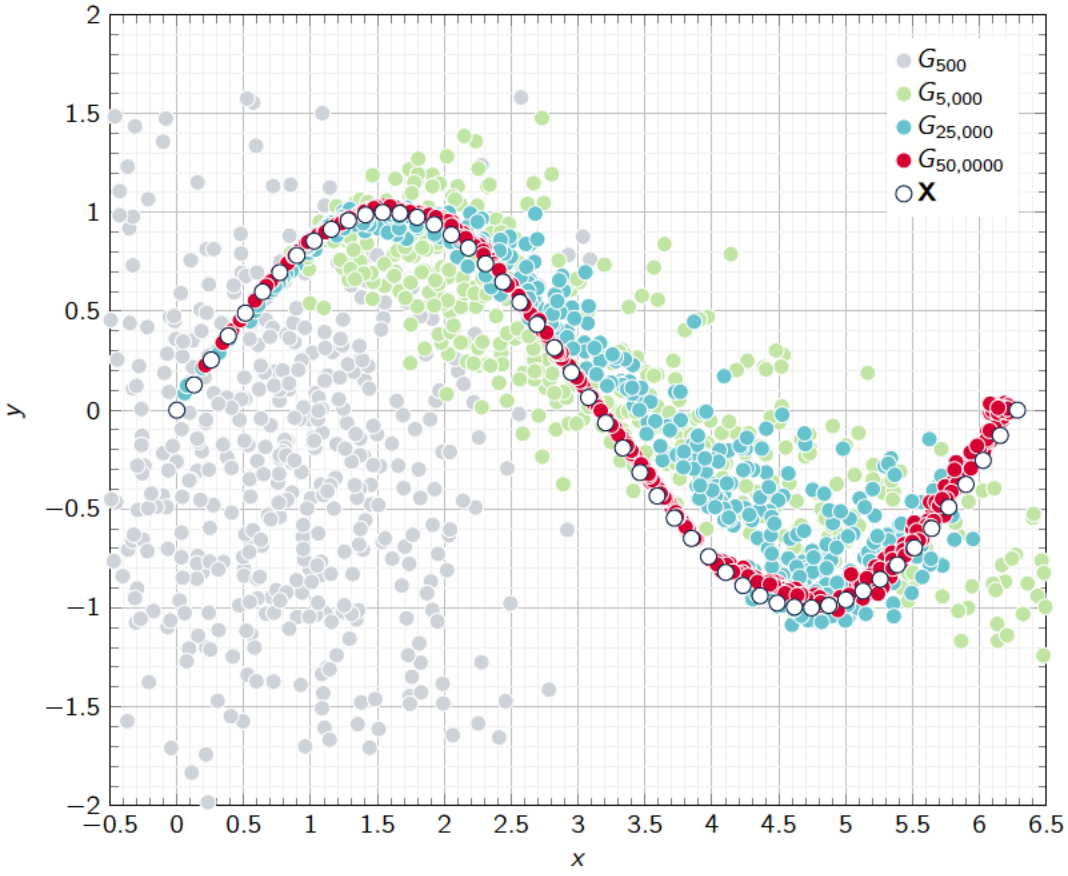


Figure 2.25: Samples generated by G after training for 500, 5,000, 25,000 and 50,000 epochs. For each plot, the trained G network was used to generate 500 samples each where **X** represents the training set used to train the GAN.

Chapter 3

Perforation of Multi Layered Targets

Chapter from published Book Chapter: F. Teixeira-Dias, S. Thompson, M. Paulino, *An artificial intelligence-based hybrid method for multi-layered armour systems*, Advanced Structured Materials — State of the Art and Future Trends in Material Modelling vol. 100, p. 323-342 (Chapter 15), A. Oechsner, H. Altenbach (eds.), Springer, 2019 [Teixeira-Dias et al., 2019]

This chapter considers the response of multi-layered targets to ballistic impact and utilises some of the key techniques presented in Chapters 1 and 2. It exists as the first of three distinct pieces of work that constitute the main body of work presented by this thesis. Specifically, an analytical energy-based method is developed based on the basic assumption that all kinetic energy is transformed during the impact between a projectile and its target [Zukas et al., 1983, Horne, 1979, Griffin, 1961]. The results of which are used to form one of two training sets used to train a MLP network as described in Section 2.2.1, where the other training set is comprised of experimental data only. Finally, FEA models are developed to validate the ML models, as described in Section 1.6.

The design of protective structures is a complex task mostly due to threat-related unknowns, such as the exact kinetic energy of the impactor and the dominant energy dissipation mechanisms. The design process is often costly and inefficient due to the number of these unknowns and to the cost of necessary steps such as laboratory testing and numerical modelling. In this chapter a hybrid method is proposed with the goal of increasing the efficiency of the design process, and consequently decreasing its cost. The method combines an energy-based analytical approach with a set of ML models. FEA and experimental results are used to train the ML models and

verify and validate the design process. The energy-based analytical method is used to establish the training set for the ML algorithms, which can then be used to find optimal configurations for the protective structure. The proposed ML model is a neural network which is trained using experimental results and analytical data, to understand the ballistic response of a specific material, and predict the residual velocity for a given impact velocity, layer thickness and material properties. Networks trained for individual layers of the armour system are then interconnected in order to predict the residual velocity of blunt projectiles perforating multi-layered composite structures. Validation tests are done on systems including single and multi-layered targets.

3.1 Introduction

Protective structures are used for a number of different purposes, ranging from protection from the environment to blast and ballistic impact. The design of protective barriers, structures and armour systems is often complex due to the number of unknowns associated with the threat, which often include the kinetic energy of the impactor (velocity and mass) and the mechanisms of energy dissipation within the protective structure or armour system. These mechanisms have been thoroughly studied and can include, for example, dissipation through plastic deformation, ductile hole growth, petalling or plugging as shown in Figure 1.1. Multi-layered structures are known to potentially increase the protection capability without significant increase in weight [Liu et al., 2018, Ali et al., 2017, Elek et al., 2005]. The design of said structures is thus very closely associated with known factors (e.g. the specific application) and unknown parameters such as those associated to the threat. The design process is often expensive and inefficient due to the number of unknowns and to the cost of involved steps such as laboratory testing and numerical modelling.

A hybrid method is proposed in this chapter to increases the efficiency of the design process while at the same time significantly decreasing its cost. The method relies on a combination of a sound analytical method, which is inherently cost-efficient, and ML models. Experimental results are used not only to inform and train the AI models but also to validate the whole design process, together with FEA. The energy-based analytical method is developed to generate a training set for the ML algorithm in order to find an optimal configuration for the protective structure, considering the most relevant energy dissipation mechanisms, and to determine perforation and residual velocity. The ML model is a neural network trained using experimental results and analytical data, with the aim of understanding the ballistic response of a specific material or set of materials, and predicting the residual velocity for given impact conditions and layer thicknesses.

3.1.1 The hybrid methodology

The proposed method relies on experimental data for the training of the ML model. FEA is used as a second validation stage, albeit not strictly necessary. Verification and validation tests are done on multiple systems, including single and multi-layered, in-contact target plates. This chapter describes the methods and presents the advantages of the proposed hybrid method over conventional FEA and experimental testing-based methodologies.

The impact of a projectile on a target can result in penetration or perforation. The former is defined as a projectile's entrance into a target without fully completing its passage through the body [Backman and Goldsmith, 1978]. This means that the striker leaves an indentation on the target, without completely perforating it. The latter describes a ballistic impact which completely pierces the target [Zukas, 1980]. In this scope, the ballistic limit velocity v_{bl} is the minimum projectile velocity that ensures perforation [Børvik et al., 1999a, Zhang and Stronge, 1996]. This velocity is a property of the armour system and is determined by a number of parameters, such as the projectile and target material properties, projectile mass and target configuration (e.g. thickness). The residual velocity is the projectile velocity after it has perforated the target. The definition of the ballistic limit velocity implies that if $v_0 = v_{bl}$ then $v_r = 0$, where v_0 is the projectile velocity just before impact and v_r is the residual velocity of the projectile. The residual velocity is zero if the target is struck by a projectile at its v_{bl} [Sikarwar et al., 2014]. The following sections detail the analytical models and AI methods used and how they are integrated in a tool that can be used to predict post-perforation residual velocities from ballistic impacts on metallic layered targets.

3.2 Analytical modelling

Protective structures and plates can be perforated in a number of different ways, which are often grouped in six main distinct perforation mechanisms as shown in Figure 1.1 [Jia et al., 2014, Woodward, 1987, Taylor, 1948, Thomson, 1955, Atkins et al., 1998, Landkof and Goldsmith, 1985]. The most common in ductile plates are ductile hole growth and plugging, shown schematically in Figures 3.1(a) and (b) [Teixeira-Dias et al., 2020]. This chapter focuses on perforation by orthogonal plugging, which occurs in finite thickness targets impacted at right angles by blunt cylindrical projectiles travelling close to or above the target's v_{bl}. The impactor forms a *plug* of target material of similar diameter to the projectile by adiabatic shearing. Adiabatic shearing is a process that occurs when a material is deformed at high strain rates. The term adiabatic refers to a process that occurs without the exchange of heat with the surrounding environment. In adiabatic shearing, the mechanical energy used to deform the material is converted into thermal energy, which increases the temperature of the material locally.

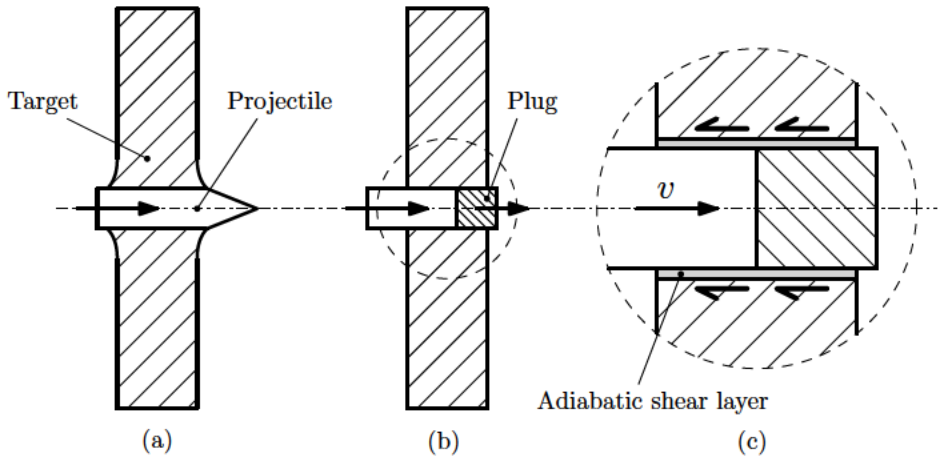


Figure 3.1: Schematic representation of the most common mechanisms of ductile plate perforation: (a) ductile hole growth, (b) plugging and (c) detail of the adiabatic shear layer, where v is the current projectile and plug velocity.

This process is commonly seen in terminal ballistics (ballistic impact), where a projectile striking a material causes it to compact and deform, resulting in the formation of a plug, more evidently when the impactor is blunt. The plug is formed by adiabatic shearing when the projectile, upon impact, pushes material in front of it, causing the material to become compacted and deformed. This deformation occurs at high strain rates, leading to a very localised increase in temperature due to the mechanical energy being converted into thermal energy. This process is known as adiabatic heating, and the resulting compacted material is known as a plug. The formation of the plug causes the material to act as a barrier, slowing down the projectile and absorbing a significant amount of energy. Adiabatic shearing can also occur in other high-strain-rate deformation processes, such as explosive forming and high-velocity impact [Oppenheim, 1996]. Plugging is initiated by plastic strains caused by high stress concentrations in a small area (and thus high stress and strain gradients), leaving the remainder of the target unaffected. Plastic strain energy is converted into heat, increasing the temperature in the shearing zone, which leads to further localisation of the plastic strain, as shown in Figure 3.1(c). In plugging this process continues until the plug completely exits the target [Krauthammer, 2008, Børvik et al., 2001b].

This section describes the main principles and derivations involved in analysing the ballistic perforation of ductile plates using energy-based principles. Conservation of energy, which is the basic principle behind these approaches, can be stated as $\sum E_{\text{in}} = \sum E_{\text{out}}$ where E_{in} and E_{out} are the input (before impact) and output (post-impact) energies of the system. The basic assumption is that all energy is transformed during the impact [Zukas et al., 1983, Horne, 1979, Griffin, 1961].

These energy-based approaches are often simplistic and thus based on a large number of geometrical, mechanical and physical assumptions and simplifications. In such models it is often assumed, for example, that thermal effects can be neglected. In the case of plugging, where adiabatic shearing is the predominant deformation mechanism, this is potentially too big an assumption. The analytical model proposed and described in this chapter tries to compensate for this by proposing an additional friction term between the projectile and the target.

The model assumes a rigid (non-deformable) projectile and is based on the relationship between stiffness and impact velocity and on the conservation of momentum. The elastic wave velocities in the projectile and target materials are c_p and c_t, respectively,

$$c_p = \sqrt{\frac{E_p}{\rho_p}} \qquad \text{and} \qquad c_t = \sqrt{\frac{(1-\nu_t)E}{\rho_t(1+\nu_t)(1-2\nu_t)}} \tag{3.1}$$

where E_p and ρ_p are the projectile's Young's modulus and density, and E_t and ρ_t are the target's Young's modulus and density, respectively. Based on the above and on the compatibility relation between the projectile and target, the contact compressive stress σ_c, dependent on the relative velocity V, is [Teixeira-Dias et al., 2020, Sikarwar et al., 2014]

$$\sigma_c = \varphi V = \left(\frac{\rho_t c_t \rho_p c_p}{\rho_t c_t + \rho_p c_p}\right) V \tag{3.2}$$

The conservation of momentum condition applied to the whole system is

$$M_p v_i = v_f M_p + v_f M_g \tag{3.3}$$

where v_f is the free impact final velocity, M_p is the mass of the projectile, M_g is the mass of the plug (the material from the target) and v_i is the projectile impact velocity. On a purely inelastic collision, the kinetic energy of the projectile is converted into deformation and heat during the impact (E_{fn}) and loss of work due to adiabatic shearing (W_n). When the projectile perforates the target there are two additional kinetic energy terms that need to be accounted for: (i) the kinetic energy of the projectile after impact, E_{kp} and (ii) the kinetic energy of the plug after impact, E_{kg}. The energy balance equation can be written as [Recht and Ipson, 1963]

$$\frac{1}{2}M_p v_i^2 = E_{fn} + W_n \frac{1}{2}M_p v_r^2 + \frac{1}{2}M_g v_r^2 \tag{3.4}$$

where v_r is the residual velocity of the projectile, assumed to be the same for the plug. The total energy fraction lost to deformation and heat during free impact, E_{fn}, must equal the difference between initial and final kinetic energies, that is,

$$E_{fn} = \frac{1}{2}\left(\frac{M_g}{M_p + M_g}\right) M_p v_i^2 \tag{3.5}$$

The work due to adiabatic shearing, W_n, is

$$W_n = \frac{1}{2}\left(\frac{M_p}{M_p + M_g}\right) M_p \left(v_{bl}\right)^2 \tag{3.6}$$

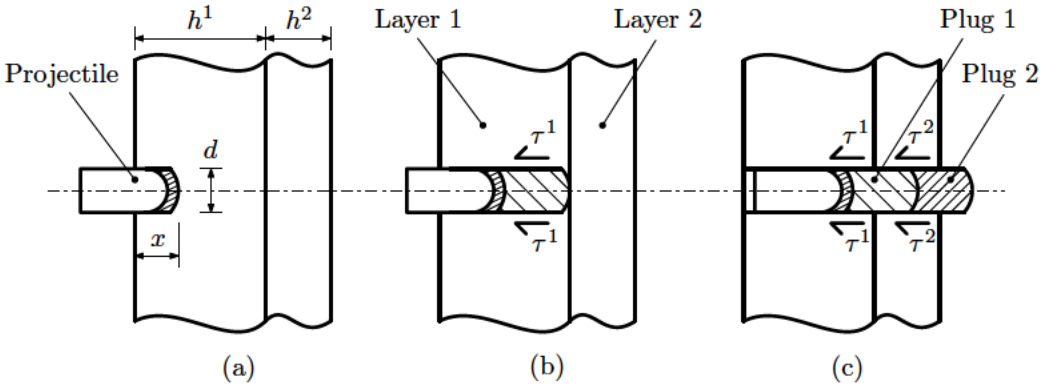


Figure 3.2: Plugging in an in-contact multi-layered target: (a) initial penetration stage, (b) formation of first plug and (c) formation of second plug [Teixeira-Dias et al., 2020].

which is derived for an initial velocity equal to the ballistic limit velocity v_{bl} of the target, that is $v_0 = v_{bl}$. W_n is insensitive to changes in velocity as long as the dynamic shear stress of the target material remains constant [Sikarwar et al., 2014].

For targets with multiple layers, the projectile is subjected to increasing compression contact stresses due to the formation of multiple plugs. Figure 3.2 shows the plug formation sequence for a multi-layered target [Teixeira-Dias et al., 2020]. By considering this incremental contact stress, which can be determined by considering the mechanical impedance resistance caused by the peripheral shear area, the energy fraction lost to deformation and heat for layer i becomes

$$E_{fn}^{i} = \frac{1}{2}\left(\frac{M_g^i \Omega^j}{M_g^i + \Omega^j}\right)\frac{\left(\sigma_c^i\right)^2 + \sigma_\tau^2}{\left(\varphi^i\right)^2} \tag{3.7}$$

where

$$\Omega^j = M_p^{i-1} + \sum_{j=1}^{i-1} M_g^j \qquad \text{and} \qquad \sigma_\tau = \frac{4h^i\tau^i}{d_p} \tag{3.8}$$

The additional energy dissipated into the peripheral shear area W_n^i can be determined by considering the average work done by the projectile in order to displace the plug of the i-th layer

$$W_n^i = \frac{1}{2}\pi d_p \left(h^i\right)^2 \tau^i \tag{3.9}$$

Assuming the residual velocity is zero ($v_r = 0$) and substituting equations 3.7 and 3.9 into equation 3.4, rearranging for the ballistic limit velocity v_{bl} and simplifying yields

$$v_{bl}^i = \frac{4h^i\tau^i\varphi^i M_g^i}{d\Omega^j}\left[1 + \sqrt{\frac{\Omega^j}{M_g^i}\left(1 + \frac{\pi d_p^3}{16\tau^i\left(\varphi^i\right)^2 M_g^i}\right)}\right] \tag{3.10}$$

This however, does not account for the velocity loss due to friction between the projectile and the hole, for each layer of the target. Based on geometrical and kinematic considerations, this velocity loss can be described by the relation

$$v_{\text{fi}} = -\left(\frac{\sigma_c^i \pi d_{\text{p}} \bar{L} \mu_{\text{k}}^i}{M_{\text{p}}}\right)\left[\frac{v^{i-1} \pm \sqrt{(v^{i-1})^2 - 2a^i h^i}}{a^i}\right] \tag{3.11}$$

where $\bar{L}$ is the friction length — the total thickness of the target or the length of the projectile, whichever is smaller. The coefficients of kinetic friction are μ_{k}^i and the deceleration of the projectile going through layer i is $a^i = (v_{\text{r}}^i - v_{\text{r}}^{i-1})/t^i$. The projectile contact time with each layer is t^i.

A generalised expression for the residual velocity can now be derived by rewriting equation 3.4 for multi-layered targets as:

$$\frac{1}{2}M_{\text{p}}^{i-1}\left(v_{\text{r}}^{i-1}\right)^2 = E_{\text{fn}}^i + W_{\text{n}}^i + \frac{1}{2}M_{\text{p}}^{i-1}\left(v_{\text{r}}^i\right)^2 + \frac{1}{2}M_{\text{g}}^i\left(v_{\text{r}}^i\right)^2 \tag{3.12}$$

Rearranging the previous equation for the residual velocity of the i-th layer v_{r}^i and including the friction term yields

$$v_{\text{r}}^i = \frac{M_{\text{p}}^{i-1}}{M_{\text{p}}^{i-1} + M_{\text{g}}^i}\sqrt{\left(v_{\text{r}}^{i-1}\right)^2 - \left(v_{\text{bl}}^i\right)^2 - v_{\text{fi}}^2} \tag{3.13}$$

3.3 ML model

3.3.1 Problem setting

This section introduces how a ML model can be used to predict the residual velocities from plugging metallic layered armour plates. This chapter compares the results from an analytical model, a numerical simulation using the finite element method and two separate neural networks; where one of these is trained on a dataset generated by the analytical model and the other entirely on experimental data collected from the literature. The experimental results published by [Børvik et al., 2003] are used as the first test case to compare the results. Each experiment used a blunt-nosed cylindrical projectile as the impactor and its respective geometry and material properties are listed in Table 3.1. The target plates were manufactured from Weldox 460 E steel and the corresponding material properties of the plate is presented in Table 3.2. The target is assumed to be fully clamped at the supports, which is a reasonable assumption as far boundary conditions are of minor importance in ballistic penetration by small mass projectiles in the ordnance velocity range when the target diameter is greater than just a few projectile diameters.

Table 3.1: Geometry and material properties of the blunt, cylindrical projectile [Børvik et al., 2003].

Diameter [mm]	Length [mm]	Density [kgm^{-3}]	Elastic Modulus [GPa]	Yield Stress [MPa]
20	80	7850	204	490

Table 3.2: Mechanical properties of the metal plate [Børvik et al., 2003].

Test [-]	Thickness [mm]	Density [kgm^{-3}]	Elastic Modulus [GPa]	Yield Stress [MPa]
1	10	7850	290	300
2	16	7850	290	300

3.3.2 AI setup

Training Sets

Two different MLP neural networks have been trained and the results compared. One network was trained only on experimental data and the other on data obtained using the analytical model described in Section 3.2. Experimental data was collected from a series of publications regarding the perforation of metal plates by blunt, steel cylindrical projectiles of the same length [Xiao et al., 2019b, Xiao et al., 2019a, Rosenberg et al., 2016, Wei et al., 2012, Børvik et al., 1999b, Huang et al., 2018, Zhou and Stronge, 2008, Rodriguez-Millan et al., 2018, Yunfei et al., 2014b, Yunfei et al., 2014a, Holmen et al., 2016, Børvik et al., 2003, Awerbuch and Bodner, 1974]. The data extracted includes key experimental parameters such as the diameter, impact velocity and residual velocity of the projectile and the thickness, modulus of elasticity, yield stress and density of the metal target plate. Material data from 2 aluminium alloys (AA-2024 and 6082-T651) and 4 steel alloys (Weldox 460, Weldox 700, Stainless 316L and Q235) were compiled to form a training set consisting of 232 samples. Evidently, the experimental data used to train the model was limited by what experiments have been performed, what materials and projectile type the researchers selected and finally what was made accessible in the literature. As a result, the collected experimental dataset is not optimal. A perfect dataset to train the neural network would include the residual velocities associated with a wider range of impact parameters, across a range of plate thicknesses and for a number of different materials. This would give a neural network the best opportunity to understand how the input parameters affect the residual velocities of the projectiles as they perforate metal plates. It is for this reason that a separate MLP network was trained on data generated by the analytical model. This represents a best case scenario where the *ideal* dataset can be replicated and presents an opportunity to assess the suitability of using neural networks to make predictions in this

domain. To that end, the analytical model was used to predict the residual velocity for 100 random impact velocities between 0 and 1000 m/s, across 10 thicknesses between 2 and 20 mm in increments of 2 mm for each of the 6 material alloys. This resulted in a total of 6000 (100 x 10 x 6) samples in the training set that was used to train the MLP model on the data from the analytical model. It should be noted that the experimental dataset collected from publications is subject to instrumental error, which is frequently defined in the region of 10% when measuring the velocity of the projectile [Børvik et al., 2003]. Random noise in the range of 0 to 10% was added to impact and residual velocities in the analytical training dataset to simulate measurement error.

Setup and training parameters

This section details the parameters and criteria defined for the training of the neural network, which was done on both the experimental and analytical datasets. Each network has six nodes in the input layer and one node in the output layer. The six input nodes allow each sample containing information regarding the diameter and impact velocity of the projectile, and the thickness, modulus of elasticity, yield stress and density of the metallic target plate to be introduced into the network. The single output node is reserved for the residual velocity of the projectile, as can be seen in Figure 3.3. The MLP network has one hidden layer consisting of 15 nodes and is activated by a ReLU activation function. The experimental dataset was split such that 70% of the samples were used for training, 15% for validation and the remaining 15% to test the performance of the model. The dataset allocated for training is used to fit the model and determine the weights between connections and biases associated with each node. The Levenberg-Marquadt (LM) algorithm was selected to train the MLP network [Ahmadian, 2016]. The LM algorithm is a combination of a loss function and an optimiser that assigns weights and biases to each node in order to best represent the function. The loss function, $E(y, t)$ depends on two parameters: the values predicted by the model y and the target values t. However, y depends on the previous layer's outputs and the current neuron's weights and activation function. Therefore it is possible to use the chain rule to differentiate $E(y, t)$ with respect to the current neurons' weights,

$$\frac{\partial E}{\partial w_{nm}} = \frac{\partial E}{\partial o_m} \frac{\partial o_m}{\partial i_m} \frac{\partial i_m}{\partial w_{nm}} \tag{3.14}$$

where w_{nm} is the weight from neuron n in the previous layer to the current neuron m. The output of an input to m are o_m and i_m, respectively. The error is then fed back through the network via back-propagation. Using this information, the algorithm adjusts the weights of each node and bias with the intention of reducing the value of the error function. The goal of neural networks is to be able to make accurate predictions on new data, which the network has not been trained on; the validation dataset is used here to expose the trained network to a new dataset and measure its performance. This allows for a second opportunity to modify

the network parameters defined before training (hyper-parameters), such as the initial weights, biases and number of hidden layers, to improve its performance on new data. Without this step, the network may be susceptible to over-fitting. This is where the error on the training set is driven down to a very small value, but when new data is presented to the network the error is large. This occurs when the network has become extremely tailored to the training examples, but has not learned to generalise to new situations and input combinations. The validation dataset is therefore useful to moderate this phenomenon. Finally, the test set is another independent dataset used to evaluate the performance of the network.

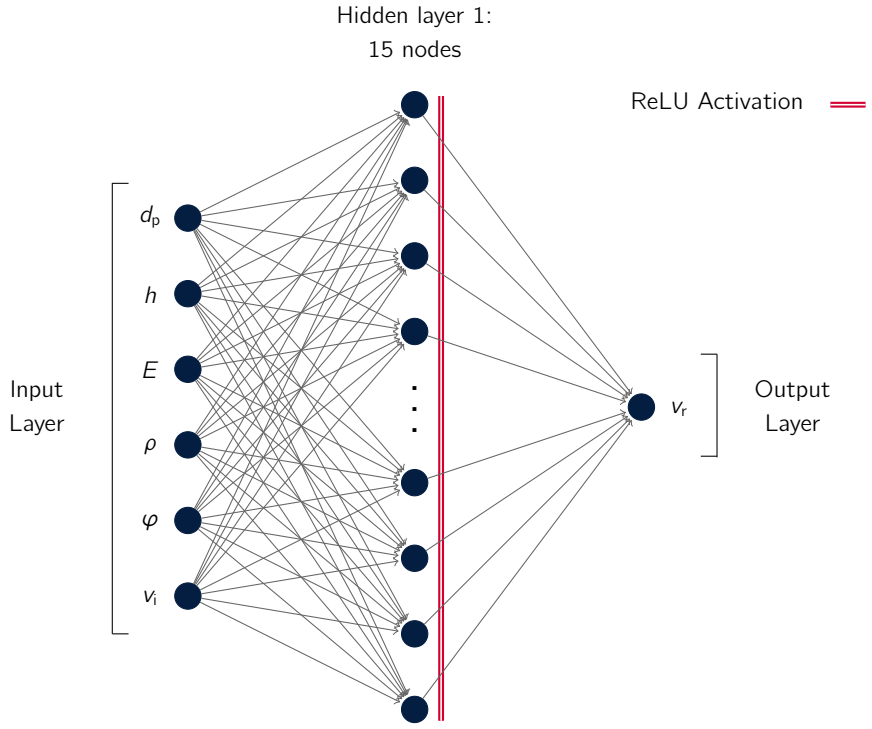


Figure 3.3: Schematic diagram of the MLP network with 15 hidden nodes, highlighting the six input nodes (projectile diameter d_p, plate thickness h, elastic modulus E, density ρ, yield constant φ and impact velocity v_0) and single output node (residual velocity v_r).

3.4 Output from the MLP model

The results in Figure 3.4 illustrate a regression plot during the training of the neural network on the experimental dataset. They show the performance of each sample during the training, validation and testing phases of training the network, and the distribution of residual velocities in the dataset, where it can be seen that the majority lie in the range of $[0, 400]$ m/s. The line

$v_r^O = v_r^T$ is plotted to show how much the predicted output from the neural network deviates from the actual target. The R-square statistical measure was used to measure how successful the fitting model was in explaining the variation of data and achieved a score of

$$R = 0.988 = 1 - \frac{\sum_{i=1}^{n}(y_i - f_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{3.15}$$

where y refers to values from the dataset, f refers to fitted values predicted by the network and $\overline{y}$ is the mean of the dataset values such that $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. The R-square value of 0.988 was computed against both the test and training data and means that the fit explains 98.8% of the total variation in the data about the average.

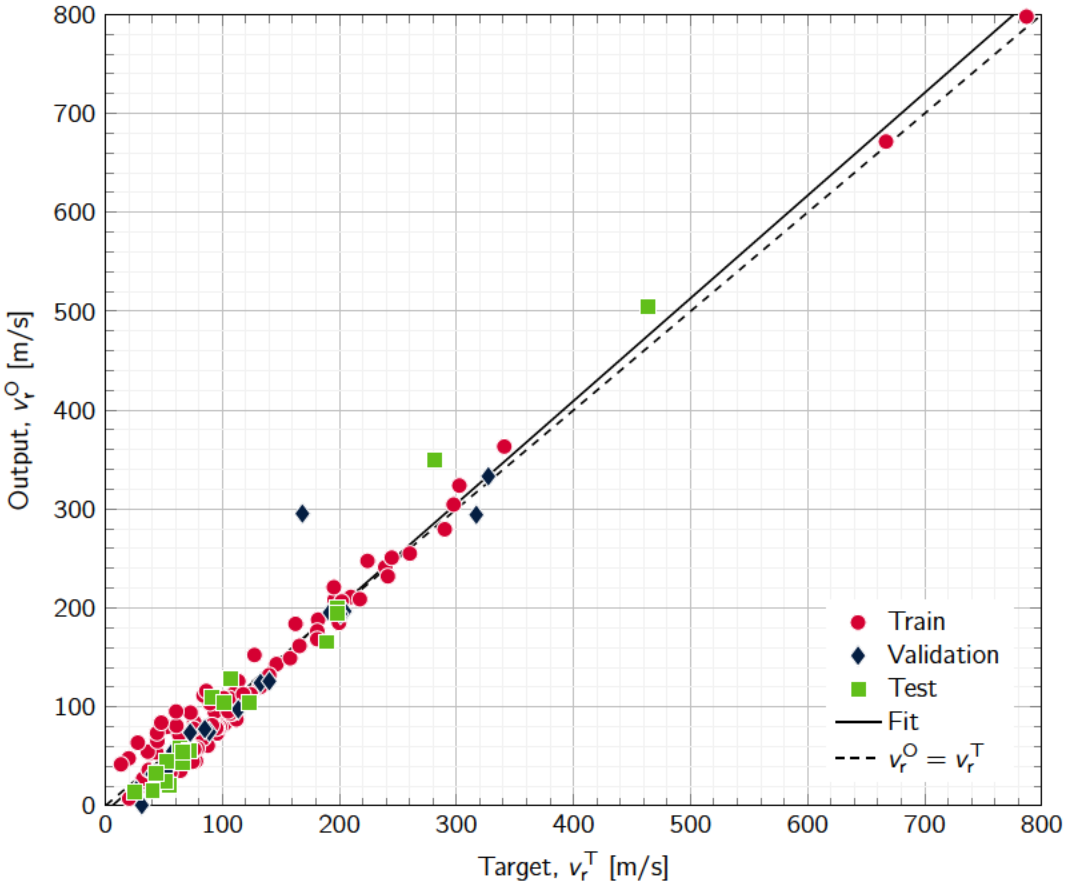


Figure 3.4: Regression plot of the performance of each sample of experimental data used during the training, validation and testing of the MLP neural network with 15 hidden layers.

Figures 3.5 and 3.6 presents the results from Figure 3.4 in a histogram such that the performance of the model during training can be visualised in more detail. The plot is arranged

into bins that represent the percentage difference between the model prediction and the expected value.

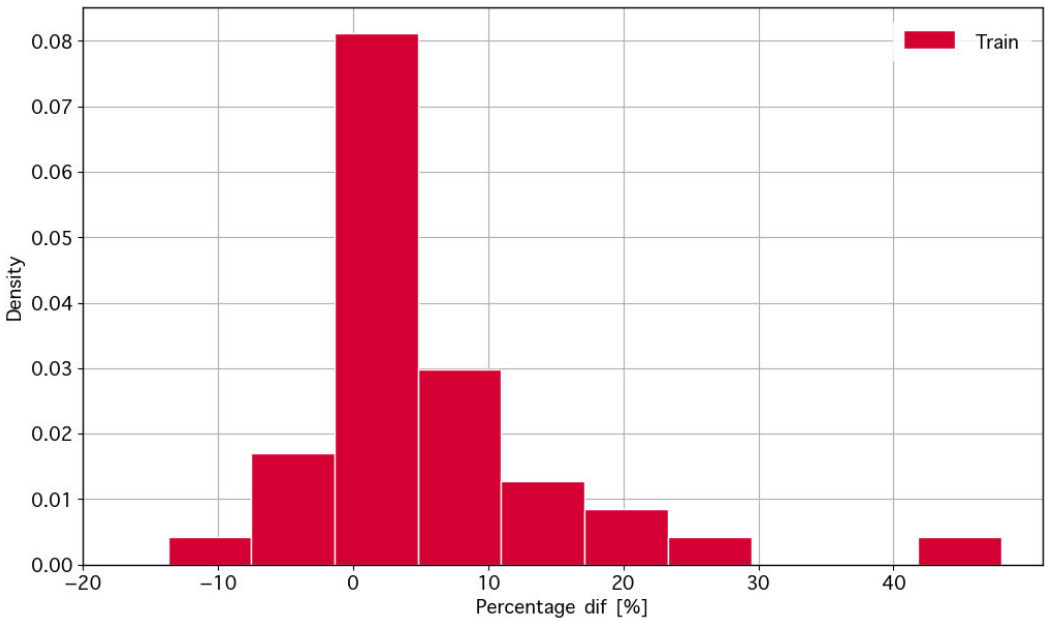


Figure 3.5: Histogram density plot indicating model performance on the combined training and validation set. The percentage difference is shown on the x axis.

Figures 3.7(a) and 3.7(b) present the results from the analytical model and the predictions from each MLP network with the experimental results published by Børvik et al. [Børvik et al., 2003] for the perforation of a blunt, cylindrical projectile perforating 10 and 16 mm Weldox 460 E plates. It should be noted that the data points from these experiments were excluded from the training set used to train the models.

The predictions from the analytical model and each neural network on Figures 3.7(a) and 3.7(b) show good agreement with the experimental data published by Børvik et al. [Børvik et al., 2003]. The analytical model and MLP network trained on the analytical dataset $\text{MLP}_{\text{a}}^{\text{N}}$, shown in Figure 3.7(a), predict a v_{bl} of 123.84 which is 25.08% lower than that found by Børvik et al. On inspection, the $\text{MLP}_{\text{a}}^{\text{N}}$ predictions match closely to that of the analytical model, this relationship is expected as the network was trained on data produced by the model, albeit with added noise to compensate for the 10% measurement error stated in Børvik et al.'s experiments [Børvik et al., 2003]. The predictions made by the $\text{MLP}_{\text{e}}^{\text{N}}$, i.e. the predictions made by the MLP network trained on the experimental dataset, predicted a projectile response that demonstrates better matching than $\text{MLP}_{\text{a}}^{\text{N}}$ with respect to both the v_{bl} and the shape of the ballistic curve. $\text{MLP}_{\text{e}}^{\text{N}}$ predicted a v_{bl} of 153.91 with an error of -6.91%. The results for the 16 mm test case show similar results. $\text{MLP}_{\text{a}}^{\text{N}}$ predicts a v_{bl} of 200.49 with an error of

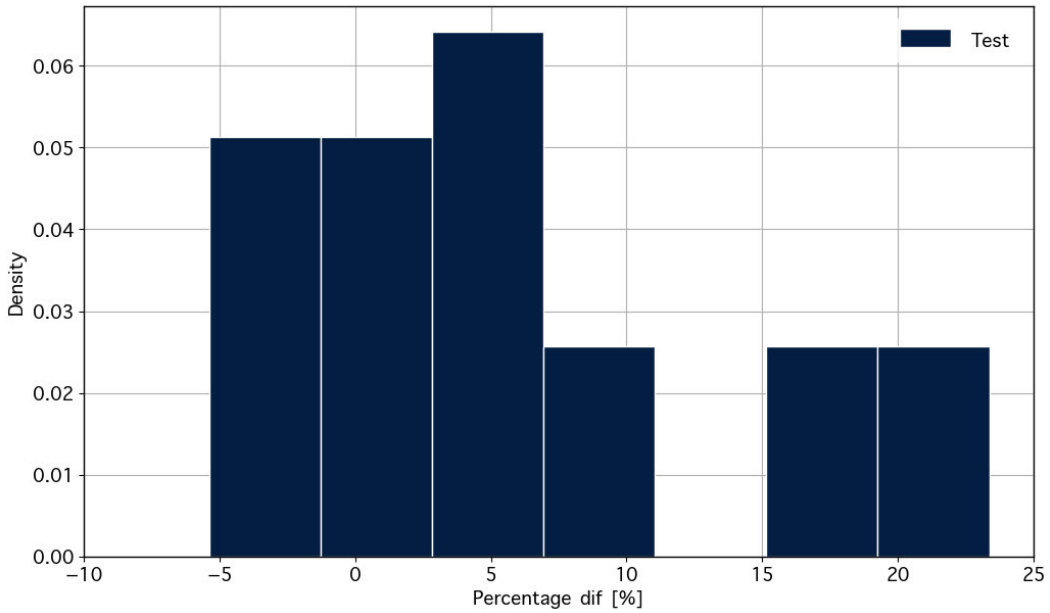


Figure 3.6: Histogram density plot indicating model performance on the test set. The percentage difference is shown on the x axis.

Table 3.3: Variance between experimental results from [Børvik et al., 2003] and the predictions from the analytical and experimental MLP models on plate thicknesses of 10 mm and 16 mm. Model predictions for thicknesses 10 and 16 are denoted as (10) and (16) respectively. $\bar{y}$ corresponds to the mean difference between the model predictions and the expected experimental results, and similarly y_{min} and y_{max} refer to the minimum and maximum differences and σ the standard deviation.

	$\text{MLP}_{\text{a}}^{\text{N}}$ (10)	$\text{MLP}_{\text{e}}^{\text{N}}$ (10)	$\text{MLP}_{\text{a}}^{\text{N}}$ (16)	$\text{MLP}_{\text{e}}^{\text{N}}$ (16)
MAE	20.40	12.15	27.11	8.92
RMSE	26.26	14.17	29.94	14.2
$\bar{y}$	8.18	−20.04	−8.65	−7.85
σ	12.13	13.35	15.06	12.57
y_{min}	−21.85	−24.86	−32.73	−37.67
y_{max}	23.35	−8.62	12.27	4.21

−15.37% and $\text{MLP}_{\text{e}}^{\text{N}}$ predicts a v_{bl} of 213.76 with an error of −9.37%. There is also lower variance in the predictions by the model trained on experimental data as it exhibits lower values for the MAE and RMSE for both the 10mm and 16mm test cases than $\text{MLP}_{\text{a}}^{\text{N}}$. Moreover, $\text{MLP}_{\text{e}}^{\text{N}}$ demonstrated that it is able to predict the v_{bl} of steel plates more accurately than the analytical model.

It should be noted that the results produced by the $\text{MLP}_{\text{e}}^{\text{N}}$ for higher velocities, i.e. greater

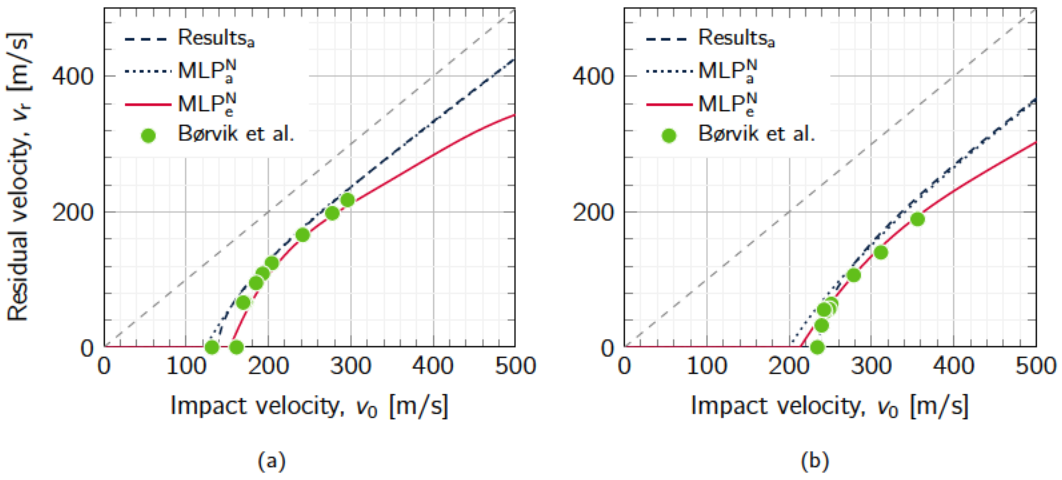


Figure 3.7: Analytical, neural network and experimental [Børvik et al., 2003] residual versus impact velocity curves of a blunt cylindrical projectile with a diameter of 20 mm perforating (a) a 10 mm and (b) a 16 mm Weldox 460 E plate, where Results$_a$ refers to the analytical model, MLP$_a^N$ are the predictions from the MLP network trained on the analytical dataset and MLP$_e^N$ are the predictions of the network trained on the experimental dataset.

Table 3.4: Comparison between the experimental ballistic limit velocities, v_{bl}, from [Børvik et al., 2003] and the MLP$_a^N$ and MLP$_e^N$. This table includes the measurement error for the experimental ballistic limit velocities and the corresponding percentage error between each model prediction for plate thicknesses of 10 and 16 mm.

	Experimental, v_{bl}	MLP$_a^N$	MLP$_e^N$
10 mm	165.3 ± 4.1	$(-25.08\%)123.84$	$(-6.91\%)153.91$
16 mm	236.9 ± 2.6	$(-15.37\%)200.49$	$(-9.77\%)213.76$

than 400 m/s, are less reliable than the predictions made at relatively lower impact velocities. This is because the predictions from the ML model are trained solely on the experimental dataset. Referring back to Figure 3.4, the majority of training samples used to train the network are within the range of $[0, 400]$ m/s. The network has therefore been exposed to more experimental data in that range and as such the respective weights and bias at each node have been optimised to best match these values. As a result, the predictions for higher velocities are less reliable and it would not be advised to use this model to study the ballistic response of materials at impact velocities above 400 m/s.

The network was additionally tested with a set of multi-layered targets. The authors were unable to find any literature regarding experimental tests on blunt, rigid, cylindrical projectiles perforating metallic plates. So to account for this, an FEA model was developed and validated against Børviks experimental results [Børvik et al., 2003]. This gave an opportunity to do a

test to assess the predictions of the MLP model on multi-layered targets.

3.5 Finite element modelling

The efficiency of the ML model described in the above paragraphs, namely when predicting residual velocities of blunt projectiles impacting layered metallic armour plates, was tested with a set of numerical examples. FEA and the hydrocode LSDYNA was therefore also used to replicate the 10 and 16 mm test cases from [Børvik et al., 2003] study on the ballistic resistance of steel plates.

3.5.1 Model information

The modelled plates and projectile were discretised with reduced integration 8-node solid elements. The density of the discretisation on the plates was set to 2 mm with a minimum aspect ratio of 0.75. The mesh density was optimised with a convergence study, where the element erosion criteria were also considered as this is strongly dependent on mesh density (i.e. element size). The plates were modelled using an elastic-plastic-kinematic material model (model `MAT_003` in LS-DYNA) and the rigid projectile was modelled using an elastic material model (model `MAT_001` in LS-DYNA) with an elastic modulus 1000 times higher than steel to minimise contact inaccuracies that arise from using a rigid material approach. Material model `MAT_001` defines an isotropic an isotropic hypoelastic material based on Hooke's law known as the Johnson-Cook model. Hooke's Law is a fundamental concept in the field of material science and solid mechanics which describes the relationship between the deformation and stress in a material. Specifically, it states that the deformation of isotropic material is proportional to the applied stress, provided the material remains within its elastic limit [Callister Jr. and Rethwisch, 2018]. For an isotropic material, Hooke's Law can be expressed mathematically as:

$$\sigma = E\epsilon \tag{3.16}$$

where σ is the stress applied to the material, E is the elastic modulus of the material and ϵ is the resulting strain or deformation of the material. The elastic modulus represents the materials ability to deform elastically in response to an applied stress, without undergoing permanent deformation. Hooke's law applies to a wide range of isotropic materials, including metals, polymers and ceramics [Ashby and Jones, 2019]. Material model `MAT_003` defines an isotropic, elasto-plastic material. Plastic kinematic material models are well suited to simulate the behaviour of materials under large deformations (such as the target plate). In plastic kinematic material models, the deformation is split into two components: elastic deformation and plastic deformation. Elastic deformation is defined by the linear relationship between stress

and strain while plastic deformation is defined by it's elastic limit and a flow rule [Meyers et al., 2006, Ashby and Jones, 2019]. The yield surface represents the boundary between elastic and plastic deformation and is a function of the stress state of the material. The flow rule determines how the material will deform plastically once it has exceeded the elastic limit. During each time step in the FEA, the material deformation is calculated as the sum of the elastic and plastic deformations. If the stress state of the material exceeds the elastic limit, then plastic deformation occurs and the material is updated using the flow rule. The updated material state is then used to calculate the subsequent stresses and strains in the next time step. [Liu et al., 2017, Dieter, 1988, He et al., 2017]

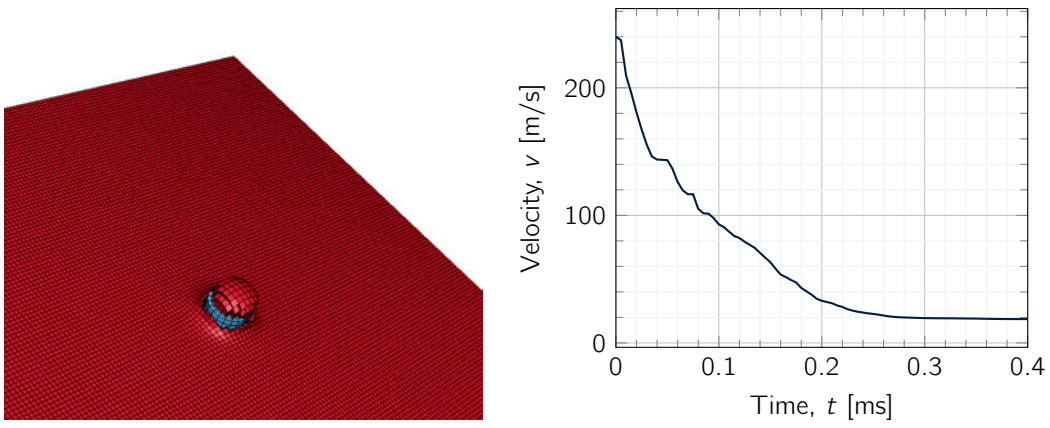
3.5.2 Single layer test cases

On a first instance the FEA models were calibrated for the ballistic limit velocity using experimental results obtained by Børvik et al. [Børvik et al., 2003]. Validation models were developed for plates with the characteristics listed in Table 3.5. The FEA models consider the same test cases as described in the previous section, where each consider a blunt cylindrical projectile impacting a Weldox 460 E plate, however, the thicknesses vary from 10 mm to 16 mm for Tests 1 and 2 respectively. Figure 3.8 shows a snapshot of FEA Test 2, for a ballistic limit velocity of 240 m/s, and the corresponding projectile velocity profile. The formation of the plug can be seen in Figure 3.8(a) and a residual velocity of approximately 18 m/s was obtained for this model, indicating that the corresponding BLV will be slightly lower than 240 m/s.

Table 3.5: Characteristics of the tests used to validate the finite element models and corresponding ballistic limit velocities; numerical ballistic limit velocity shown in brackets. Material properties can be found in Table 3.2.

Test	Plate Material	Thickness [mm]	Ballistic Limit velocity (FEA) [m/s]	Number of elements
1	Weldox 460 E	10	165.6 (166)	135,000
2	Weldox 460 E	16	236.6 (240)	225,000

A number of additional numerical tests were ran to further validate the hybrid-method here proposed, which combines analytical and experimental results to train a predictive neural network. The models, which were defined with specifications outside the set of results used for training the ML model, are listed in Table 3.6. All these tests were done at impact velocities above the ballistic limit velocity and the residual velocity was used as the validation parameter. As can be seen from these results, there is very good agreement with the experimental results with the ML model always lower than 5.1%. Discrepancies become significant (as high as 35.2% for Test V4) when comparing with the FEA results, however, this is believed to be related to the method used to model adiabatic shearing and the formation of the plug. The finite element



(a) (b)

Figure 3.8: Snapshots of finite element analysis results for Test 2 (see Table 3.5) for a ballistic limit velocity of 240 m/s: (a) deformed plate showing the formation of the plug and (b) velocity profile for the projectile.

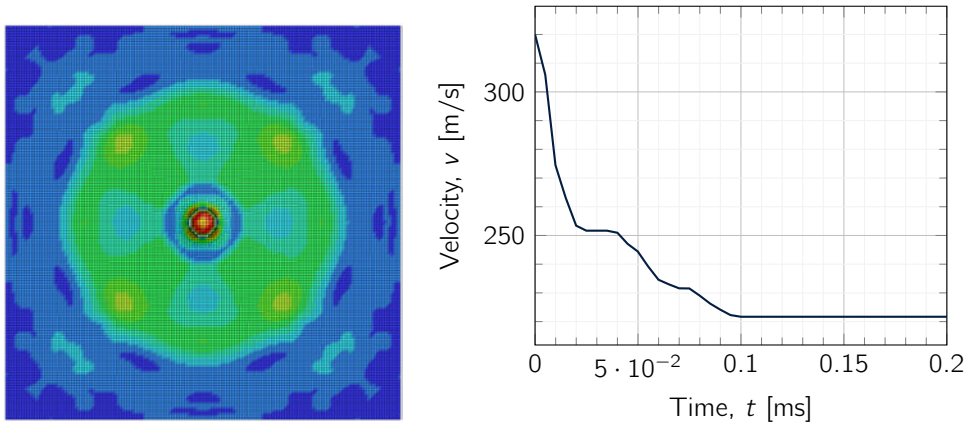
analyses use element deletion for this, which is known to be inaccurate and potentially deviating from mass conservation of the system. Figure 3.9 shows a snapshot of validation Test V3, for an impact velocity of 320 m/s and the corresponding projectile velocity profile.

Table 3.6: Specifications, results and comparison of the tests used to validate the ML algorithm. Tests V1 to V6 correspond to impacts on the Weldox 460 E plate.

Test ID	h [mm]	v_0 [m/s]	Experimental v_r [m/s]	FEA v_r [m/s]	ML v_r [m/s]	FEA/ML [%]	Exp/ML [%]
V1	10	220.0	143.1	126.6	136.2	7.0 (−)	5.1 (+)
V2	10	280.0	201.7	184.1	196.6	6.4 (−)	2.6 (+)
V3	10	320.0	234.6	221.7	226.4	2.1 (−)	3.6 (+)
V4	16	260.0	83.5	108.7	79.9	35.2 (+)	4.5 (+)
V5	16	280.0	111.9	143.3	108.7	31.8 (+)	2.9 (+)
V6	16	320.0	153.3	189.4	157.6	20.2 (+)	2.7 (−)

3.5.3 Multi-layered targets

To further validate the AI method, finite element analyses were run on a multi-layer target. Once configured, each FEA simulation took 30 minutes to complete. In this test, the multi-layer configuration consists of a 6 mm 45 Steel Plate and a 6 mm Q235 Steel Plate where the impactor makes first contact with the plate made from 45 Steel. The impactor is a blunt,



(a) (b)

Figure 3.9: Snapshots of finite element analysis results for validation Tests V3 (see Table 3.6) for an impact velocity of 320 m/s: (a) deformed plate showing the initial stages of the formation of the plug and von Mises stresses, and (b) velocity profile for the projectile.

cylindrical projectile with a diameter of 12.67 mm. The material properties of the projectile and the plates are listed in Table 3.7.

Table 3.7: Material properties of the projectile and each of the 6mm Steel plates. In the arrangement the 45 Steel plate is struck by the projectile first.

Material Properties	Projectile	45 Steel Plate	Q235 Steel Plate
Young's modulus [GPa]	204.0	190.0	190.0
Density [kg/m^3]	7850.0	7850.0	7850.0
Poisson's ratio [-]	0.3	0.3	0.3
Yield Stress [MPa]	–	552.0	500.0
Tangent modulus [MPa]	–	450.0	450.0

The FE model was used to determine the projectile's residual velocity across a range of impact velocities between 0 and 700 m/s, the results of which are presented in Table 3.8. It can be seen that for $v_i < 200$ no perforation occurs. This means that the projectile did not possess the kinetic energy required to perforate the back of the second plate. For $190 < v_i < 215$, partial perforation is observed. This means that the projectile perforated the back of the second plate but became stuck as the kinetic energy it possessed was not enough to overcome the friction between the projectile and the plate. For $v_i > 210$, complete perforation is observed and the projectile completely passes through the multi-layer target.

The analytical model and MLP_e^N were also used to predict the response of the same multi-layer configuration. The results of which are presented in Figure 3.10. It can be seen that the results from both the analytical model and MLP_e^N match closely with that of the FE model. The v_{bl} for the MLP_e^N and the analytical model is 226.2 and 220.3 m/s respectively, both of which are within 10% of that predicted by the FEA model.

Table 3.8: Residual velocity v_r of a blunt cylindrical projectile impacting the multi-layer target as predicted by the FEA, .

v_0 [m/s]	FEA [m/s]	MLP_e^N [m/s]	Analytical Model [m/s]
100	0.0	0	0
150	0.0	0	0
180	0.0	0	0
190	0.0	0	0
200	0.0	0	0
210	0.0	0	0
215	5.2	0	0
218	29.1	0	0
220	48.1	0	0
230	80.6	12.56	49,.44
245	81.7	57.74	80.93
250	125.0	70.12	87.68
280	170.0	127.89	130.9
300	192.0	157.14	154.33
500	404.2	364.94	340.59
600	502.1	458.99	423.53

3.6 Concluding remarks

This chapter presents a new approach that looks to predict residual velocities from impacts on monolithic and multi-layered metallic ductile targets. The proposed method utilised a combination of experimental, analytical and numerical methods, and a set of MLP models. The aim of which was to test for any equivalence between the results from numerical and machine learning models. If an equivalence can be observed, then ML could be a viable alternative to numerical models given the vast reduction in computational time and domain knowledge required to obtain results. The experimental and analytical results were used to train two separate MLP neural networks, while the FE analyses were used primarily to validate results. Excellent agreement was obtained in most cases of impact on monolithic targets, with error

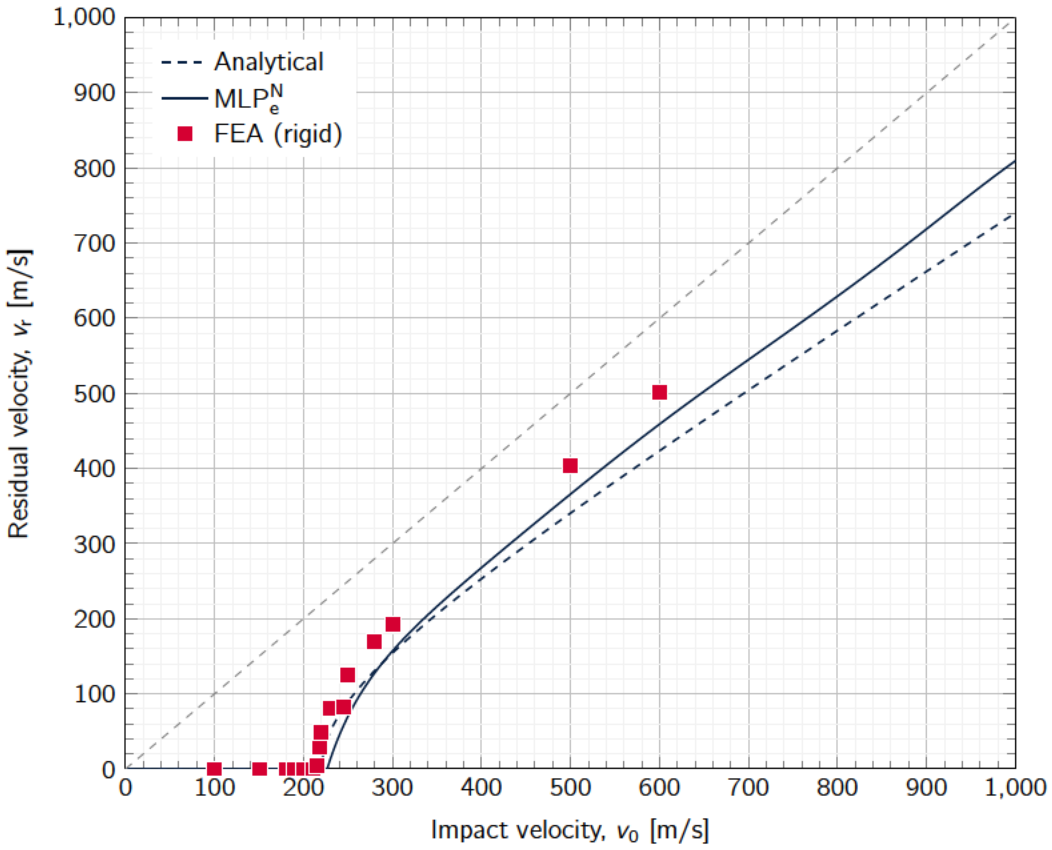
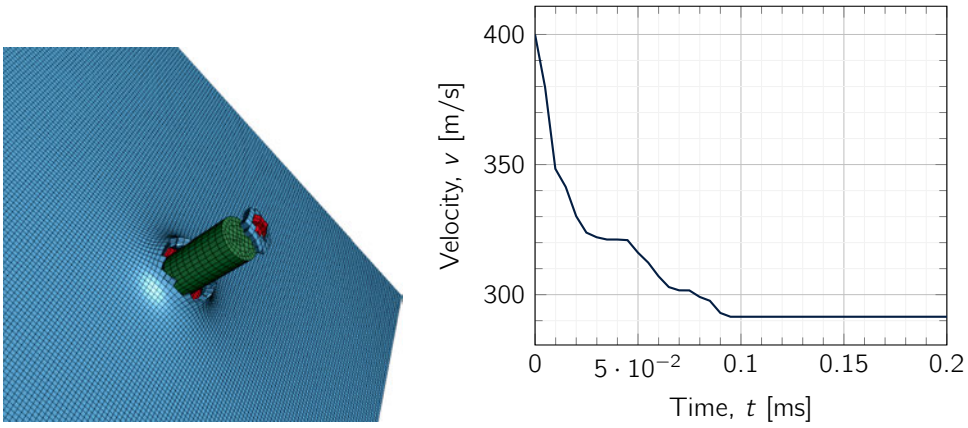


Figure 3.10: Plot indicating the relationship between impact velocity v_i and the residual velocity v_r of a 12.67 mm blunt, cylindrical projectile impacting a 12 mm bi-layer plate where the first plate is made from 45 Steel with a thickness of 6 mm and the second plate is Q235 Steel that also has a thickness of 6 mm. Predictions are displayed from the analytical model, MLP$_e^N$ and the FEA model.

levels under 10%. The proposed hybrid approach loses efficiency, however, when the dominant energy dissipation mechanisms are not considered or are difficult to model in the analytical and numerical approaches. These energy dissipation mechanisms include high levels of plastic deformation on the projectile or target deformation modes other than plugging.

The MLP network architecture was trained on an analytical dataset, formed by results from the energy-based analytical model, and an experimental dataset. The experimental dataset was compiled by sourcing experimental results from the literature. A key difficulty with using experimental data to train ML models is that often the data is sparse. A ML model benefits from a comprehensive training set. For this application, a comprehensive training set would contain ballistic samples across a wide variety of impact velocities across a wide variety of target plate thicknesses. This would give the MLP model the best opportunity to learn patterns



(a) (b)

Figure 3.11: Impact on a multi-layered target: (a) detail of the interaction between the projectile and target for Test V8 (rigid projectile) showing the formation of the double plug, and (b) the corresponding projectile velocity profile.

in the data and learn to determine the relationship between how the input parameters (material properties of target and projectile and the loading case) influence the output parameter (residual velocity). Comprehensive training sets such as this however, are difficult to find in the literature due to the large costs associated with experimental testing. The next chapter aims to address this problem, and presents a novel approach to supplement sparse ballistic datasets by utilising a different form of ML model known as the GAN that was detailed in Section 2.8.

Chapter 4

Ballistic response of armour plates using GANs

Chapter from published Journal Article: S. Thompson, F. Teixeira-Dias, M. Paulino, A. Hamilton *Ballistic response of armour plates using Generative Adversarial Networks*, Defence Technology, 2021 [Thompson et al., 2021]

This chapter expands upon the work from the previous chapter but instead considers the problem of sparse, ballistic datasets. It proposes a unique solution by supplementing the sparse datasets by using a recent class of machine learning system known as the Generative Adversarial Network (GAN). GANs are regularly used to generate additional examples for image datasets [Goodfellow et al., 2014, Radford et al., 2016, Karras et al., 2018a] and traditionally they require large, comprehensive training sets to yield favourable results. In the case of image generation, it can take 50,000 to 100,000 training images to train a high-quality GAN [Karras et al., 2018b, Brock et al., 2019]. In this study however, it is demonstrated that for regression tasks it is possible to train a GAN on a much smaller training set and generate additional samples representative of the data present in the training set. To that end, this study investigates the possibility of training a GAN directly on sparse, ballistic datasets. The intention being that the trained GAN can then be used to generate new ballistic samples as opposed to performing additional destructive experiments. In this chapter a GAN network architecture is proposed, and tested and trained on three separate ballistic data sets. The trained networks were able to successfully produce ballistic curves with an overall RMSE of between 10 and 20% and predicted the v_{bl} in each case with an error of less than 5%. The results demonstrate that it is possible to train generative networks on a limited number of ballistic samples and use the trained network to generate new samples representative of the data that it was trained on.

This study spotlights the benefits that generative networks can bring to ballistic applications and provides an alternative to expensive testing during the early stages of the design process.

4.1 Generative Adversarial Networks

For a machine learning application, a perfect ballistic training set would include observations from experiments performed across a range of impact velocities with different materials and different armour system configurations (e.g. different plate thicknesses). However, due to the destructive nature of these tests, there is a high cost associated with ballistic experiments. This chapter proposes a solution to this problem using a different neural network, a GAN, that has been specifically developed to supplement ballistic data sets. A GAN is a recent class of machine learning system that has the ability to generate entirely new data and make predictions through unsupervised learning [Salimans et al., 2016]. Given a training set, this technique learns to generate entirely new data with the same statistical representation as the training set. This allows additional ballistic samples to be generated using the model as opposed to performing additional destructive tests.

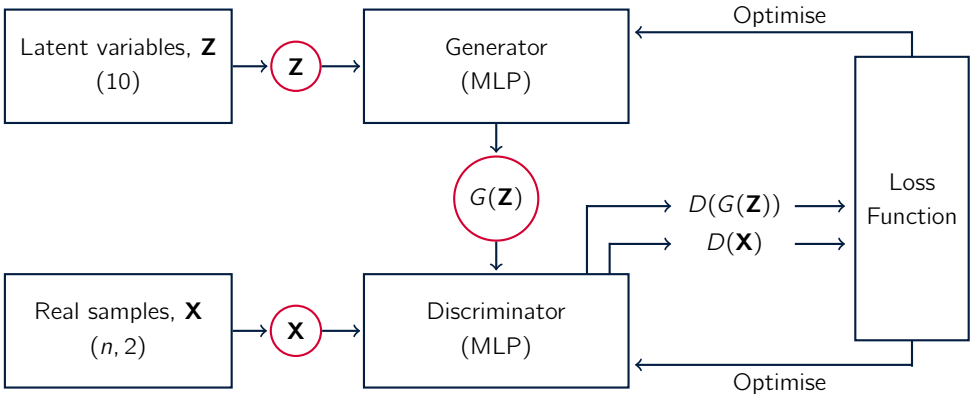


Figure 4.1: Schematic diagram of a Generative Adversarial Network (GAN).

The base architecture of a GAN is composed of two neural networks: a discriminator and a generator, as shown in Figure 4.1. The discriminator D is set up to maximise the probability of assigning the correct labels to real and fake samples. Meanwhile, the generator G is trained to fool the discriminator with synthesised data [Chen et al., 2020]. In other words, D and G play the following two-player minimax game with value function $V(G, D)$ [Goodfellow et al., 2014].

$$\min_G \max_D V(D, G) = E_X \log D(\mathbf{X}) + E_Z \log[1 - D(G(\mathbf{Z}))] \tag{4.1}$$

where $\mathbf{X}$ is the input to D from the training set, $\mathbf{Z}$ is a vector of latent values input to G, E_X is the expected value over all real data instances, $D(\mathbf{X})$ is the discriminator's estimate of the

probability that real data instance $\mathbf{X}$ is real, E_Z is the expected value over all random inputs to the generator and $D(G(\mathbf{Z}))$ is the discriminator's estimate of the probability that a fake instance is real. The primary goal of G is to fool D and produce samples that D believes come from the training set. The primary goal of D is to assign a label of 0 to generated samples, indicating a fake, and a label of 1 to true samples, that is, samples that came from the training set. The training procedure for G is to maximise the probability of D making a mistake, i.e. an incorrect classification. In the space of arbitrary functions G and D, a unique solution exists, with G able to reproduce data with the same distribution as the training set and the output from $D \approx 0.5$ for all samples, ultimately indicating that the discriminator can no longer differentiate between the training data and data generated by G.

4.1.1 Training sets

Ballistic testing is a standards-based process where materials are tested to determined whether they meet protection, safety and performance criteria. In this study, ballistic experiments refer to the V_{50} ballistic test, where projectiles are fired at higher velocities to determine a key design parameter known as the ballistic limit velocity v_{bl}. In this scope, the v_{bl} is the minimum projectile velocity that ensures perforation [Børvik et al., 1999a, Zhang and Stronge, 1996]. This velocity is a property of the threat-armour system and is determined by a number of parameters, such as the projectile and target material properties, projectile mass and target configuration (e.g. thickness). This study aims to reduce the costs associated with ballistic experiments by minimising the number of experiments performed and supplementing the data set by using the GAN model instead. It discusses and compares the results from three separate GAN models trained on three separate training sets. An appropriate training set is required such that the discriminator can learn the distribution of the data. In this case, the generative networks are trained to generate new samples of ballistic data. It is therefore important to prepare an appropriate data set that can be used for this purpose.

In order to test the capabilities of this method, the samples generated by the GAN are compared with the equivalent values produced by the Lambert and Jonas relation, as detailed in Section 1.5. This was selected for two reasons: (i) the Lambert and Jonas equation can be used to generate the training sets for each test case to effectively test proof of concept, and (ii) it provides a useful metric through which to directly compare the results of the GAN predictions. Three different test cases were considered, leading to three different training sets to test the performance of the proposed GAN architecture. Each training set is a $(n \times 2)$ array where n is the number of samples in the training set, the first column corresponds to the impact velocity v_{i} and the second column to the residual velocity v_{r}. The first test case, Case 1, represents a best case scenario and consists of 100 logarithmically spaced data points to maximise the number of points around v_{bl}. These points represent v_{i} with the corresponding v_{r}

calculated using equation 1.1 and the parameters listed in Table 4.1. The Case 1 training set, shown in Figure 4.2(a), covers a wide range of impact velocities, including impacts for velocities both above and below v_{bl}, providing the learning algorithm with the best chance to learn an idealised representation of the data. Case 2, shown in Figure 4.2(b), consists of 50 randomly generated data points in the residual velocity test range, which in this case is $[0, 600]$ m/s. The training set used for Case 2 is therefore less structured and less dense than Case 1. The final test case is the most realistic and attempts to replicate experimental data in the form that is typically found published in the literature [Xiao et al., 2019b, Xiao et al., 2019a, Rosenberg et al., 2016, Wei et al., 2012, Huang et al., 2018, Zhou and Stronge, 2008, Rodriguez-Millan et al., 2018] and is shown in Figure 4.2(c). 10 points were generated at random in the residual velocity test range $[0, 600]$ m/s and similarly arranged into a (10×2) matrix. A $\pm 10\%$ artificial noise was added to v_r to mimic experimental measurement error, ensuring that the training data no longer sits on the Lambert ballistic curve, as can be seen in Figure 4.2.

Table 4.1: Lambert parameters used to create the training sets and test each GAN model.

Lambert parameter	Value
a	1
p	3
v_{bl}	100

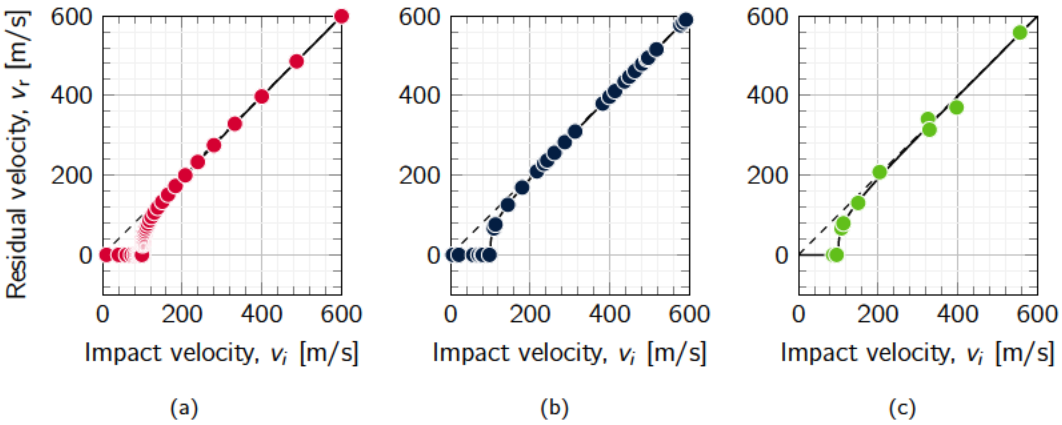


Figure 4.2: Training sets used to train each GAN model: (a) Case 1, (b) Case 2, and (c) Case 3.

Each of the training sets were normalised between the range $[0, 1]$ before training commenced. It should be noted that the numeric values in the dataset are transformed via a common scale, without distorting differences in the ranges of values or losing any information. After training, the output from G is scaled back to the original values by using the same object to undo the

transformation. For each test case, the trained GAN model generates 100 samples and the curve is fitted in accordance with the Lambert and Jonas model with respect to parameters a, p and v_{bl} to obtain a new set of Lambert parameters specific to the GAN-generated data and curve. Due to the stochastic nature of generative models, this is performed 100 times for each test case and the average values for each parameter are recorded and compared.

4.1.2 Model architecture

MLP networks were used to create the G and D networks as they are well equipped to deal with regression tasks and the adversarial modelling framework is straight forward to apply when both models are MLPs [Goodfellow et al., 2014]. The discriminator D takes an instance from either the generator or training set as input, and outputs a classification prediction as to whether the sample is real or fake — it is a binary classification problem. The discriminator network is an MLP network with 5 fully connected hidden layers, with 25, 15, 15, 5 and 1 nodes in each layer. The model minimises the following binary cross entropy loss function:

$$L_{CE}(Y, \hat{y}) = -\frac{1}{n}\sum_{i=0}^{n}\left[Y_i \log(\hat{y}_i) + (1 - Y_i)\log\left(1 - \hat{y}_i\right)\right] \tag{4.2}$$

where $\hat{y}$ is the predicted value, Y is the true value and L_{CE} is the binary cross-entropy loss. The Adam version [Kingma and Ba, 2014] of the stochastic gradient descent method was selected to update model parameters during training and a LReLU activation function was selected to moderate the output from each of the hidden layers in the Discriminator model [Xu et al., 2015]. Details regarding the Adam algorithm are presented in Section 2.4.1. The LReLU moderates the output by allowing positive inputs to pass through unchanged such that $f(x) = x$ for $x > 0$ and for negative inputs LReLU allows a shallow non-zero negative gradient. This is contrary to the typical ReLU activation function where for negative input values the output is zero such that $f(x) = \max\{0, x\}$ [Maas et al., 2013]. Finally, the output layer of the discriminator model passes through a Sigmoid activation function to moderate the output values in the range $[0, 1]$ [Han and Moraga, 1995].

The generator model G takes an input $\mathbf{Z}$ from the latent space and generates a new sample. A latent variable is a hidden or unobserved variable, and the latent space is a multi-dimensional vector space of these variables. The latent input is a random vector that serves as the input to G. This vector's dimensionality is a hyperparameter of the model. G takes this latent input vector and uses it to generate a synthetic data point, with the goal of making it indistinguishable from data from the training set. In contrast, the network weights are learned parameters that are part of the generator and discriminator network themselves. These weights are updated during training to improve the performance. While the latent input vector and network weights are both important components of the model, they serve difference purposes. The latent input

vector is used to generate diverse synthetic data by introducing randomness into the network, while network weights are optimised to minimise the difference between the synthetic and real data distributions. Model G uses 10 latent variables as input to the model that exists as a 10 element vector of Gaussian random numbers. The majority of GANs published in the literature focus on using GANs for image generation and analysis. The intentions of the model proposed in this study differ from the literature in the sense that the training sets used to train the models are smaller and contain fewer data, and secondly the desired output is less complex (training sets considered consist of 100, 50 and 10 samples with two features as opposed to commonly used MNIST [Lecun et al., 1998] and CIFAR10 [Krizhevsky and Hinton, 2009] image datasets that contain $60,000$ and $10,000$ image samples respectively with 28/32 features depending on the size of the image). As a result, the authors considered a smaller range of latent input sizes between 2 and 30 and found that a latent input of 10 resulted in a stable model that was able to capture the key features of ballistic curves consistently for different datasets. G has two fully connected hidden layers with 11 hidden nodes and is activated with a Rectified Linear Unit (ReLU) activation function, as detailed in Section 2.3.3. The weights associated with each node are initialised with uniform scaling between 1 and 10. The output layer has two nodes for the two desired elements v_i and v_r, and a linear activation function is used to output real values. The network architectures for both models are listed in Table 4.2.

Table 4.2: Architecture of the G and D networks: n is the number of entries in the respective training set. FC refers to a fully connected layer, FC 25 to a fully connected layer with 25 nodes, and LReLU to a Leaky ReLU activation function.

Discriminator	**Generator**
Input: $(n, 2)$	**Input**: Latent Dim 10
FC 25, LReLU	FC 11, ReLU
FC 15, LReLU	FC 11, ReLU
FC 15, LReLU	FC 2, Linear
FC 5, LReLU	
FC 1, Sigmoid	
Output: $D(\mathbf{X})$ or $D(G(\mathbf{Z}))$	**Outputs**: v_i, v_r

4.1.3 Training algorithm

The machine learning algorithm was written in Python3, using TensorFlow's high-level Keras API for building and training deep learning models. The training process takes place over 1 million iterations and a single iteration is completed once the machine learning algorithm has completed an entire pass through the training data set (one epoch). Within each iteration, the trainable parameters of both the D and G networks are updated alternatively; first the D

network is trained and then the G network. D is simply a classifier that learns to distinguish real data (label = 1) from fake data generated by G (label = 0). During training D therefore aims to output a value of 1 when evaluating real samples and a value of 0 when evaluating fake samples and updates its trainable parameters accordingly via the binary cross entropy loss function and adam optimiser. D can therefore be trained alone and is trained on both data from the training set and also from fake data generated by G. The D network is therefore trained on two epochs of data per iteration and, if left uncorrected, would train at a faster rate than G, thus giving D an advantage and therefore affecting the competitive nature of the adversarial process. In order to correct this, the entire sample fed to D is split in half to form a batch. This is done randomly at each iteration to ensure that the total data that D is trained on per iteration is the same as that of G. The G network is trained entirely on the performance of D and its trainable parameters are optimised in accordance with its output $D(G(\mathbf{Z}))$.

A single iteration of training is complete once the trainable parameters of both the D and G networks have been updated, twice for D (on real and fake samples) and once for G. First, real samples from the training set $\mathbf{X}$ are prepared and passed into D to obtain $D(\mathbf{X})$. A target label of 1 is concatenated with the training data to identify them as real samples. The output $D(\mathbf{X})$, that is its prediction as to whether the sample is real or fake, then passes into the loss function and the trainable parameters of D are optimised with the intention of returning a value closer to the target label of 1 for the next real sample it receives. Similarly, D is also trained on fake data from G in the form of $G(\mathbf{Z})$ and is concatenated with a target label of 0 to identify them as fake samples. $D(G(\mathbf{Z}))$ then passes through the loss function and the trainable parameters of the D network are updated with the intention of $D(G(\mathbf{Z}))$ returning a value closer to the target label 0 for future fake samples. The goal of G however, is to generate samples that D believes to have come from the original training set X, therefore the output $D(G(\mathbf{Z}))$ is used to train the G network. This defines the zero-sum adversarial relationship between the two models. As the training of G depends on D, the network cannot be trained in isolation. A combined model is therefore required to form the GAN. The GAN is a sequential model that stacks both the G and D networks such that G receives the latent Gaussian vector $\mathbf{Z}$ as input and can directly feed its output into D. To that end, $G(\mathbf{Z})$ is concatenated with a target label of 1 (indicating a real sample) and the output $D(G(\mathbf{Z}))$ passes through the loss function and the trainable parameters of G are updated with the intention of $D(G(\mathbf{Z}))$ returning a value closer to the target label of 1. It should be noted that the binary cross entropy loss function and adam optimiser are also used to train the combined GAN model and that whilst G is being trained, the nodes within each layer of the D network are frozen and cannot be updated; this prevents D from being over-trained on fake examples. The complete training algorithm of the GAN is shown in Table 4.3.

Table 4.3: Training algorithm for the Generative Adversarial Network.

Input: Training Set $\mathbf{X}$, latent input $\mathbf{Z} \sim P_{\mathrm{Z}}$, epochs T, learning rate ϵ, batch size m
Output: Generated samples $G(\mathbf{Z})$

start
Define: D and G MLP networks
Define: Combined sequential model GAN with D and G
Initialise: Trainable parameters $\boldsymbol{\theta}$ of D and G
Freeze: Trainable parameters of D in combined GAN model

for t = 1 : T **do**
Collect samples $\{\mathbf{X}_i\}_{i=1}^{m}$ from training set $\mathbf{X}$, label = 1
Generate samples $\{G(\mathbf{Z})\}_{i=1}^{m}$ from latent Gaussian distribution P_{Z}, label = 0
Train D on real samples $\{X\}_{i=1}^{m}$ and update D, label = 1
Train D on fake samples $\{G(\mathbf{Z})\}_{i=1}^{m}$ and update D, label = 0
Invert class labels of $\{G(\mathbf{Z})\}_{i=1}^{m}$ from 0 to 1
Train combined GAN model on $\{G(\mathbf{Z})\}_{i=1}^{m}$ with inverted labels and update G
end for
end

4.1.4 Model evaluation

This section details the process adopted to evaluate the success of the proposed models. A total of three separate GAN networks have been trained and named in reference to the training set that they were trained on (e.g. GAN1 refers to the generative network trained on the first training set etc.) In this study, the expected values are known as the model is aiming to recreate results representative of the Lambert model. Each model was trained for a total of 1 million iterations and the performance of the GAN was evaluated at every $1,000$ iterations as the model in its current form was used to generate 100 samples. The RMSE was calculated by taking each v_{r} predicted by the GAN model and computing the difference with the expected value from the Lambert model for the respective v_{i}. The non-linear least squares method was used at each evaluation point to fit the Lambert equation to the generated data such that Lambert parameters specific to the generated model at that point during training could be obtained. The percentage difference of these parameters with the expected parameters, listed in Table 4.1, was computed and plotted to show how the accuracy of the samples generated by the GAN changes throughout training. This results in four parameters, $a_{\%}$, $p_{\%}$, $v_{\mathrm{bl}\%}$ and the RMSE, that are being monitored during training to evaluate the performance of the model. It should be noted that for the model's intended application this would not be the case and instead the generative network would simply be presented with experimental samples that would

form the training set. It would have no true reference and as such it can be difficult to know the optimal point in time to terminate the training.

GANs can prove difficult to train and a lot of research is ongoing to improve the convergence of generative networks [Kodali et al., 2017, Nowozin et al., 2016, Theis et al., 2015]. Often the most meaningful way to interpret the success of the GAN is with visual interpretation. It would therefore be recommended to regularly use the GAN during training to generate ballistic samples as an additional qualitative measure to evaluate the training process. That being said, for this application 1 million iterations afforded the learning algorithm enough opportunity to consistently optimise the parameters of the network on different training sets, without an unreasonable compromise in computational cost. Once the generative network is trained, it is important to study the quality of the output to determine the success of the model. However, due to the stochastic nature of the GAN and the latent Gaussian input to the network, the output varies. A statistical analysis for each of the generative networks to gain further insight into its output. For each of the GAN networks, the $1,000,000$th iteration of the model was used to generate 100 samples of data and parameters $a_{\%}$, $p_{\%}$, $v_{\mathrm{bl}\%}$ and the RMSE were once again calculated. This was done $1,000$ times and each of the four parameters were stored at each iteration in a $(4 \times 1,000)$ array, the results of which are discussed in the next section.

4.2 Results from GAN model

A total of three separate GAN networks have been trained on three separate training sets. The variation of parameters $a_{\%}$, $p_{\%}$, $v_{\mathrm{bl}\%}$ and the RMSE are plotted against training time in iterations alongside a 100 sample output of the final model in Figure 4.3. This is done for each of the three networks, with Figures 4.3(a) and (b) referring to GAN1, Figures 4.3(c) and (d) to GAN2 and finally Figures 4.3(e) and (f) to GAN3. On inspection, it can be seen that the samples generated by each of the GAN networks match the shape of the Lambert curve. In the case of GAN1, which was trained on the first training set, Figure 4.3(a), there is noticeable improvement in the accuracy of the network as training progresses. Both the RMSE and the respective Lambert parameter errors determined via the fitting model decrease with training. This is a clear indication that the model has learned from the training set and is now able to generate new samples that are representative of that initial data set. This observation is enforced when looking at the samples generated by GAN1 in Figure 4.3(b). The generated samples appear to come from the same distribution as the training set and display close matching both before and after the ballistic limit.

The training set used to train GAN1 was the most structured and comprehensive of the three training sets and thus provide optimal conditions for the learning algorithm to optimise its respective trainable parameters. Figures 4.3(c) and (d) show the equivalent results for GAN2.

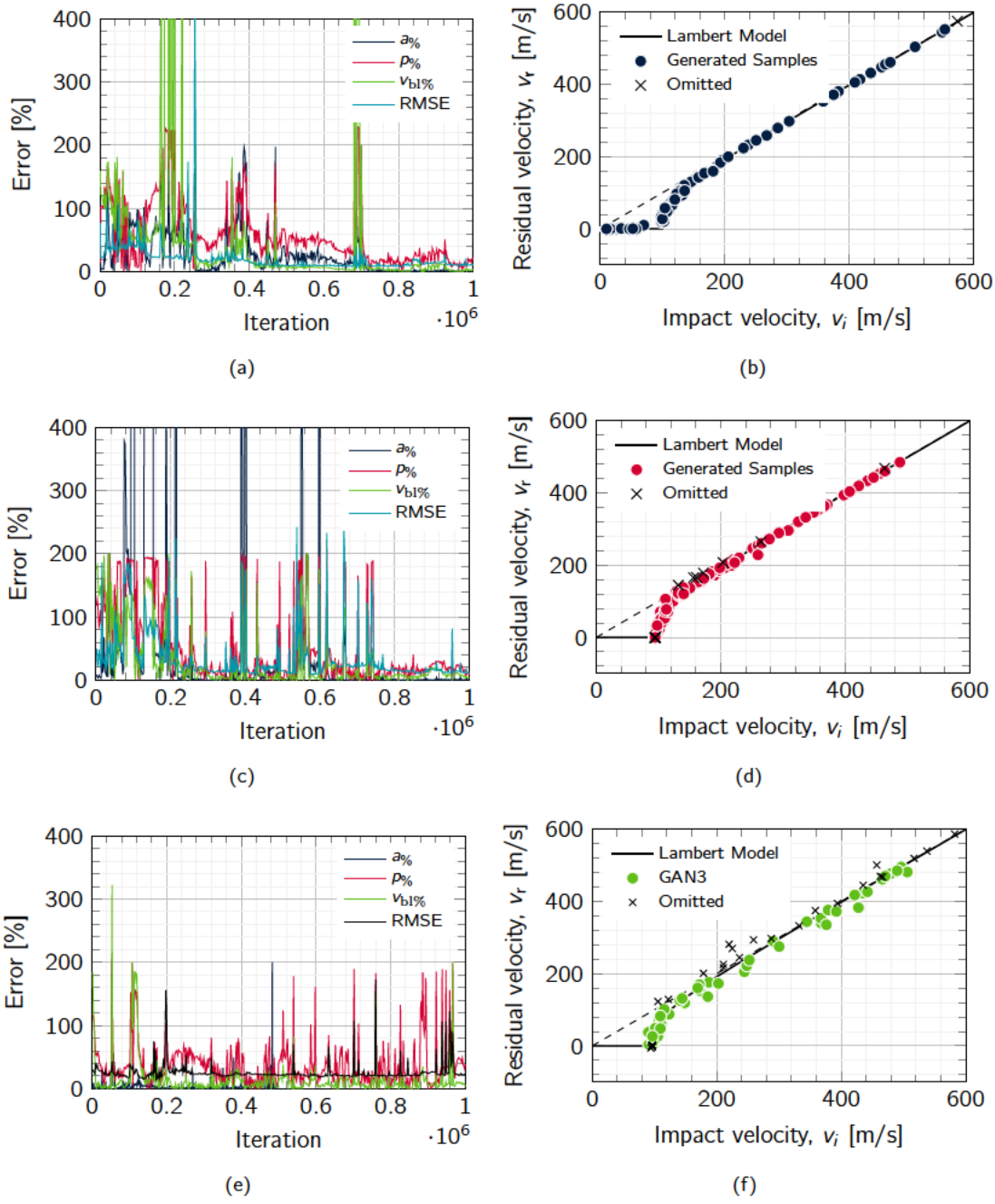


Figure 4.3: Plots (a), (c) and (e) show the error of the model during training including the RMSE and the percentage difference of fitted Lambert parameters with expected values. $a_{\%}$ is the percentage error of the fitted parameter specific to the GAN network at that point in training with the true Lambert value a, etc. Plots (b), (d) and (f) show a 100 sample output from GAN1, GAN2 and GAN3 after 1 million iterations, respectively, where the respective D and G models were trained with $\epsilon = 0.0035$. Plot includes omitted values that meet either of the elimination criteria: (i) if $v_i < v_{bl}$ then v_r must not be less than zero, and (ii) $v_r > v_i$.

Once again it can be seen that over time the errors of each of the Lambert parameters decrease with number of iterations as the model learns and the output of GAN2 begins to stabilise. Earlier in the training process large error spikes are visible where the output from the model is highly inaccurate. These correspond to points where optimisation was unsuccessful, the loss function would therefore return larger values and the trainable parameters of the network would then be updated more rigorously in order to reduce the loss and improve accuracy. The final iteration of GAN2 was used to generate the 100 samples shown in Figure 4.3(d). The generated samples are also consistent with those of the Lambert model and representative of the training set that it was trained on. Samples generated by GAN2 demonstrate good matching with the Lambert model past the ballistic limit velocity. However, unlike GAN1, GAN2 does not generate samples below the ballistic limit. The training set used to train the GAN1 model represents an optimal training set and consists of 100 samples logarithmically spaced around the ballistic limit velocity. This provides the GAN model with training data from the entire impact range of $[0, 600]$ m/s and subsequently the best opportunity to learn the behaviour of the correct ballistic response. The training set used to train GAN2 however, consists of 50 samples with x values randomly selected between the impact range of $[0, 600]$ m/s and the corresponding y values calculated via the Lambert equation. In comparison to the first training set, this data is unstructured and no priority has been made to organise the data around the ballistic limit. Of the 50 samples in the training set, only 7 exist below the ballistic limit. During training, it is likely that GAN2 was optimised such that it converged to a local minima where a solution was found where D believes that samples generated by G belong to the training set, but without including the additional feature that represents the horizontal line at $v_r = 0$ for $v_i < v_{bl}$. It could also be that 7 samples beneath the ballistic limit is insufficient to correctly learn that feature of the ballistic curve, however more targeted research would have to be conducted to affirm that conclusion. For the ballistic application however, this is not an issue as residual velocities beneath the ballistic limit velocity are, by definition, equal to zero.

The results of GAN3 are shown in Figures 4.3(e) and (f). GAN3 was trained on the training set that was the least comprehensive and most representative of data collected via experiments. This training set contains much fewer samples and, unlike training sets 1 and 2, was tainted with additional noise of up to 10%, to mimic experimental measurement errors. Figure 4.3(e) shows the variation of parameter accuracy during training. This model did not converge as successfully as with GAN1 and GAN2, and appears to be less stable demonstrating more spikes in error throughout training. Figure 4.3(f) shows the 100 samples from the final iteration of GAN3 and it can be seen that once again the samples follow the shape of the ballistic curve. The samples generated by GAN3 have a larger spread than those generated by GAN1 and GAN2, but this is consistent with the tainted training set that it was trained on. GAN3 does not demonstrate samples for impact velocities v_i between the range $[0, v_{bl}]$, however this is expected as samples within that range are not present in the training set. A comparison of the coefficients generated by the final iteration of each GAN network can be found in Table 4.4.

Table 4.4: Comparison of fitted models with the Lambert coefficients. Values shown are average values taken from 1000 runs where the parameters are determined from the 1,000,000th iteration model generating 100 samples of ballistic data.

Coefficients	Lambert	GAN1	GAN2	GAN3
a	1.0	1.01 (+0.1%)	1.00 (+0.00%)	1.00 (+0.00%)
p	3.0	2.44 (−20.58%)	3.11 (+3.60%)	4.89 (+47.90%)
v_{bl} [m/s]	100.0	96.82 (−3.23%)	99.27 (−0.73%)	104.95 (+4.83%)
RMSE	–	10.48%	11.99%	22.44%
Omitted samples	–	2.31	11.70	27.46

The results in Table 4.4 compare the average Lambert coefficients determined by curve fitting the samples generated by each of the GAN networks $1,000$ times with the baseline Lambert parameters listed in Table 4.1. The results show that all of the GAN networks performed well with respect to parameter a with the percentage error in each case $< 0.1\%$. GAN2 was the most successful in regards to p with an average error of 3.6%, outperforming GAN1 and GAN3 which had errors of -20.58% and 47.90%, respectively. This remains true for the v_{bl} case as GAN2 also produced the lowest average errors of -0.73%, GAN1 predicted the v_{bl} with an error of 3.23% and finally GAN3 with an error of 4.83%. This metric is particularly useful as the v_{bl} is an important parameter when determining the ballistic response of materials and for all GAN networks the predictive error was $< 5\%$. Figure 4.4 shows the influence that parameters a, p and v_{bl} have on the ballistic curve. The values of the parameters used in the Lambert equation are listed in Table 4.1 and Figures 4.4(a), (b) and (c) demonstrates the effect of changing parameters a, p and v_{bl}, respectively.

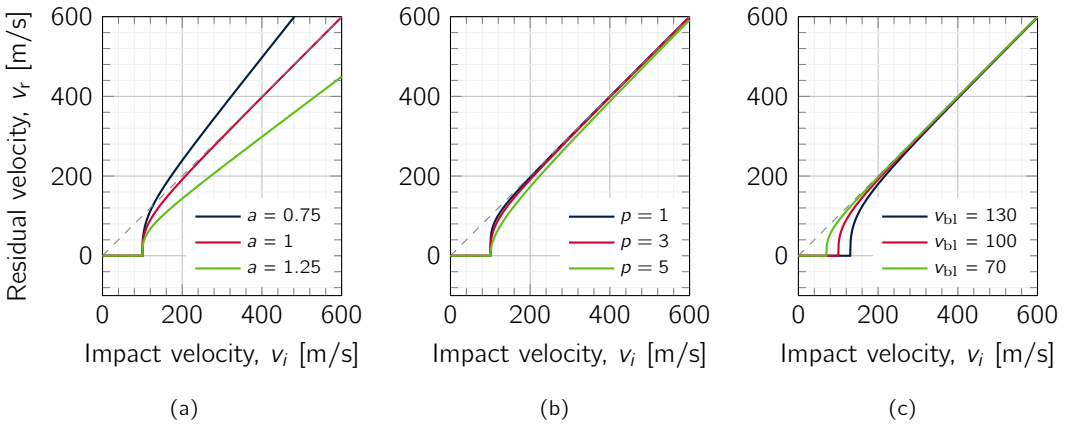


Figure 4.4: The effect that modifying the Lambert parameters has on the shape of its ballistic curve. Default Lambert parameters used are $a = 1$, $p = 3$ and $v_{\text{bl}} = 100$ m/s, plot (a) shows the effect of changing parameter a, plot (b) shows the effect of varying parameter p and (c) the variation of v_{bl}.

Figure 4.4 (a) demonstrates that decreasing a raises the profile of the curve and increasing it does the opposite. $a = 1$ ensures that the ballistic curve asymptotically approaches the line $y = x$ (slope 1) which is consistent with energy conservation. Parameter p controls the gradient of the curve at impact velocities $> v_{bl}$; it can be seen that increasing p results in a curve that takes longer to approach the line $y = x$ from v_{bl} as a point of reference faster whereas decreasing it does the opposite. As expected, it can be seen in Figure 4.4 (c) that altering v_{bl} shifts the position of the ballistic limit velocity on the x-axis. This plot is important as the large errors found in Table 4.4 for parameter p can be misleading, as despite a large percentage difference to the actual Lambert parameter, the shape of the ballistic curve does not differ as much as might be expected. A better metric to consider the overall accuracy of the results is the RMSE, where GAN1 was the most accurate with an overall error of 10.48% and GAN3 was the least successful with an overall RMSE of 22.44%. The notable increase in error between GANs 1 and 2 with GAN3 is expected since GAN3 was trained on a reduced training set that had been tainted with additional noise. Omitted samples on Figures 4.3 (b), (d) and (f) are plotted on the ballistic curve for completeness. These generated samples are unrealistic and were omitted for meeting one of two elimination criteria: the first refers to generated samples with a negative residual velocity, which would indicate that the projectile did not possess the kinetic energy necessary to perforate the target plate and as such rebounded. Ballistic experiments published in the literature typically do not record the velocity of the rebounded projectile and label such occurrences with $v_r = 0$ [Børvik et al., 2003, Mohammad et al., 2020, Ali et al., 2017]. Therefore the first elimination criteria is to remove samples where $v_r < 0$. The second elimination criteria refers to cases where $v_r > v_i$, which is a violation of conservation of energy. It is physically impossible for a projectile to perforate a plate and gain kinetic energy. Instead, kinetic energy from the projectile would be lost and transformed into heat energy and strain energy within the plate to facilitate its deformation — v_r can never exceed v_i and as such samples that meet that criteria are also eliminated. It should also be noted that the omitted values were still used when calculating the results shown in Table 4.4.

Due to the stochastic nature of the GAN output, it is important to analyse the output statistically to gain further insight into the results. The parameters a, p and v_{bl} are determined by curve fitting the 100 generated samples to the Lambert model to obtain parameters specific to the GAN. This was done 1,000 times and the values for each GAN were stored to create four matrices of dimension (1000×4) that corresponds to the output, note this was also done for the RMSE. A Kernel Density Estimation (KDE) plot was used to estimate the probability density function of each parameter and the results are shown in Figure 4.5. Statistical parameters from each of the GAN networks are shown in Table 4.5. Figure 4.5(a) displays the predicted values of parameter a by each of the GAN networks. It can be seen that each of the models performed well as the densities for each GAN are high and the peaks of the KDE curve are close to the expected Lambert value $a = 1$. From the plot it is clear that the output from GAN3 has a wider distribution than that of GAN1 and GAN2. From Figure 4.5(b) it is clear that the

output from GAN2 is most consistent with the expected values as the peak of the KDE plot lies closely to the Lambert prediction. The overall distribution from GAN1 is narrower than GAN2. However, it consistently under predicts p. The output of GAN3 with regard to this metric follows a much larger distribution with no clear peak. The average value of p shown in Table 4.4 was 4.89 yet in some cases the fitted p value was much higher with some notable outliers. Figure 4.5(c) shows the results for parameter v_{bl} and each of the models performed well on this metric with the peaks of each KDE plot lying close to that of the true value of 100 with the traditional bell-shaped curve. Once again the distribution from GAN3 is much broader and GAN2's output appears non-normal demonstrating a bi-modal distribution the main peak before the v_{bl} and the secondary peak after. Finally Figure 4.5(d) shows the calculated values for the RMSE where it can be seen that GAN1 was the most successful model with a narrow distribution and the tallest peak. GAN2 has a slightly shallower peak at a higher RMSE and a wider distribution. Finally GAN3 was the worst performing network with the shallowest peak and a wider distribution than the other models.

Table 4.5: Standard deviation (σ), minimum (y_{min}) and maximum (y_{max}) values associated with parameters a, p, v_{bl} and RMSE for generative networks GAN1, GAN2 and GAN3. Values are fitted from 100 samples generated by each GAN network 1,000 times and values are taken from stored array.

		a	p	v_{bl}	RMSE, %
GAN1	σ	0.00	0.17	2.13	1.21
	y_{min}	1.00	1.68	81.72	6.07
	y_{max}	1.03	2.99	101.48	15.13
	$\bar{y}$	1.01	2.44	96.82	10.48
GAN2	σ	0.00	0.38	2.94	1.46
	y_{min}	0.99	2.51	95.45	7.75
	y_{max}	1.01	5.83	114.57	18.04
	$\bar{y}$	1.00	3.114	99.27	11.99
GAN3	σ	0.01	11.03	11.89	3.16
	y_{min}	0.96	1.93	94.68	13.78
	y_{max}	1.05	110.82	180.00	33.99
	$\bar{y}$	1.00	4.89	104.95	22.44

4.3 Concluding remarks

In this chapter, a novel approach was proposed to generate realistic ballistic samples by training GANs on ballistic data directly. Three separate GAN networks each trained on a unique dataset created using the Lambert and Jonas ballistic model, as detailed in Section 1.5. In total, there

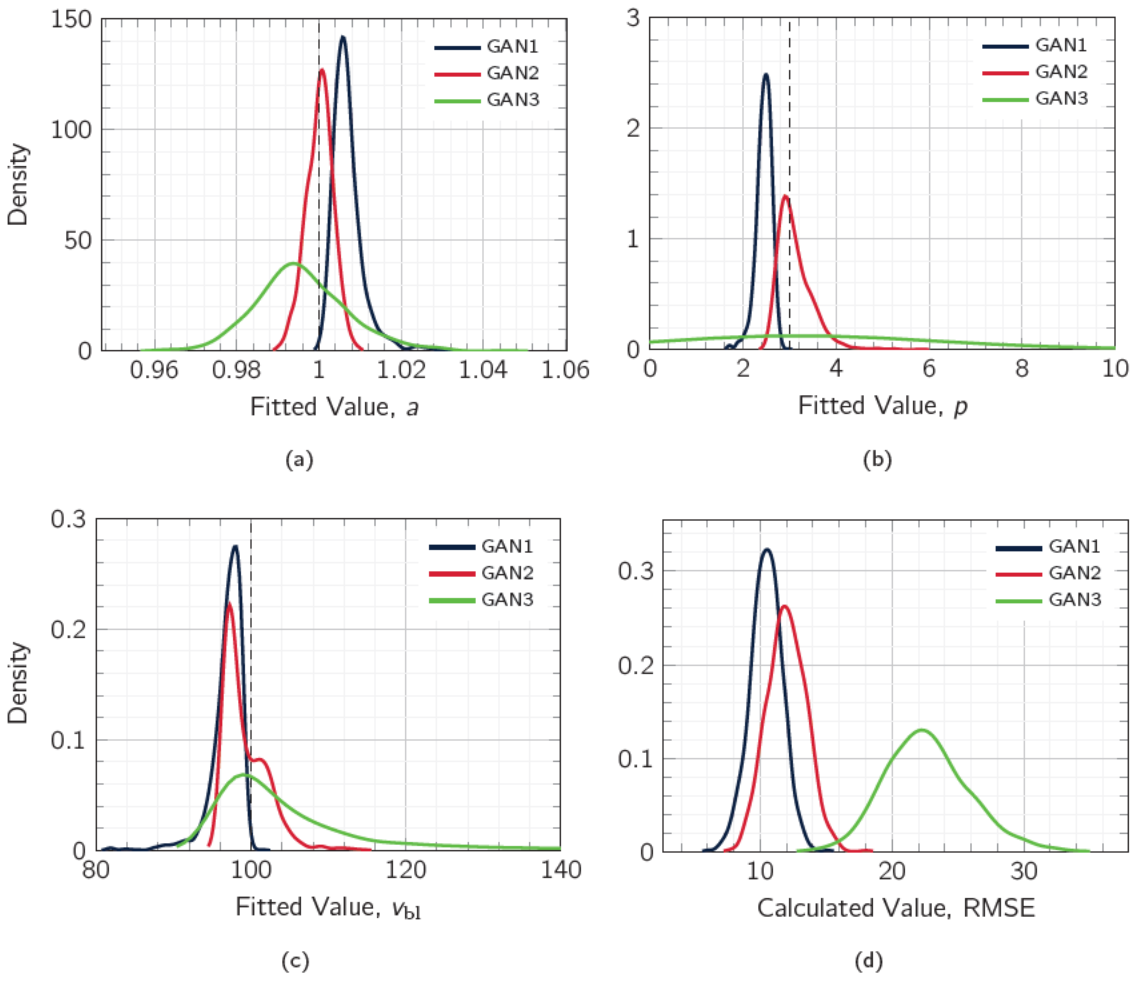


Figure 4.5: The Kernel Density Estimation (KDE) plot for each Lambert parameter predicted by each of the GAN networks. Plot (a) compares the distributions of a, (b) compares the distributions of p, (c) compares the v_{bl} and finally (d) the RMSE.

were three training sets of degrading structural quality containing 100, 50 and 10 samples in training sets 1, 2 and 3 respectively; where training set 3 was afflicted with an additional 10% noise to mimic that of measurement error. The GAN models were trained for a total of 1 million iterations and in each case, the trained networks were capable of generating additional samples that on inspection matched the shape of the Lambert ballistic curve. The models were successfully able to reproduce samples representative of the training set that it was trained on. The model was used to generate 100 ballistic samples 1,000 times such that a thorough analysis of the output can be performed. The GAN models predicted the v_{bl} with an error of -3.23, -0.73 and 4.83% with an average RMSE of 10.48, 11.99 and 22.44%, respectively.

The GAN architecture and training parameters proposed resulted in a stable training process for each of the ballistic test cases considered. The output from each of the GAN models improved with training and did not suffer from common issues such as non-convergence and mode collapse and thus additional stability precautions were not applied.

Chapter 5

Predictions on multi-class ballistic datasets using cGANs

Chapter from published Journal Article: S. Thompson, F. Teixeira-Dias, M. Paulino, A. Hamilton *Predictions on multi-class ballistic datasets using conditional Generative Adversarial Networks*, Defence Technology, 2022 [Thompson et al., 2022]

This chapter builds upon the GAN study detailed previously, but instead implements a variant known as the conditional-GAN (cGAN). The cGAN addresses a limitation of conventional GAN networks where there is limited control over the output. A cGAN network can be conditioned on additional information during training such as class labels in order to govern its output. In the space of material characterisation campaigns, experiments are performed to determine the ballistic response of a specific material plate across a range of thicknesses. Such experimental campaigns are a rich topic of research in the literature and include work by [Børvik et al., 2003, Børvik et al., 2005, Kristoffersen et al., 2020a, Yunfei et al., 2014b]. This presents an interesting opportunity to investigate whether machine learning, specifically cGAN networks, can be used as an alternative method to predict the ballistic response of plates across multiple classes. In this instance, a class could correspond to a different plate thickness. This chapter investigates whether it is possible to train a cGAN network on a multi-class ballistic dataset, where each class corresponds to a different ballistic curve (analagous to that of different thickness experiments in an armour plate characterisation campaign), and determine whether the trained cGAN network can generate new class-specific ballistic samples. Unlike the conventional GAN, the trained cGAN model can also be used to make predictions for classes that do not appear within the training set. This is an important distinction as it dramatically improves the number of useful application cases within the field of ballistic research. For example, a trained cGAN network

might be trained on a ballistic dataset with class labels $1-5$ where a class label of 1 refers to ballistic samples associated with a thickness of 10 mm, class label of 2 refers to a thickness of 20 mm, etc. To that end, non-integer class labels could be passed through the cGAN network to predict the ballistic response for thicknesses (intermediate classes) that are not present in the dataset, without the cost of performing more experiments.

In this study, a single MLP cGAN architecture is trained on a multi-class ballistic training set consisting of 10 classes labelled $0-9$ where each class refers to a ballistic curve with a different ballistic limit velocity, v_{bl}. A total of 5 models were trained on training sets consisting of 5, 10, 15, 20 and 25 ballistic samples within each class. For integer class labels $0-9$, all cGAN models successfully predicted the v_{bl} with an error of less than 4.12%. It was also found that for non-integer class labels between $0-9$ the v_{bl} predictions were similar despite not explicitly appearing in the training set. Moreover, each of the cGAN models was challenged to generate new samples for class labels that exist beyond the scope of the training set for class labels between $9-20$. It was found that four of the models were able to predict the v_{bl} with an error of less than 1.5% in all cases. This study showcases the capability of the cGAN model to learn directly from a multi-class ballistic dataset and generate additional samples representative of that data for classes that did not appear explicitly in the training set.

5.1 Methodology

The objective of this study is to develop a new cGAN architecture capable of generating new predictions representative of those originated by ballistic impact experiments. The main outcomes from this study are:

1. To propose and demonstrate that a trained cGAN can be used to supplement existing ballistic datasets.
2. To use the cGAN model to make predictions on key engineering properties such as the ballistic limit velocity v_{bl}
3. To make predictions for intermediate classes that have not explicitly been performed and do not exist in the training data.

A cGAN is developed and trained on a multi-class dataset. The goal of the cGAN is to generate new ballistic data across a range of classes where each class could refer to an important material parameter such as the target plate's thickness. [Thompson et al., 2021] demonstrated how a GAN model can be used to supplement ballistic datasets and make predictions of key experimental parameters such as the ballistic limit velocity. A drawback with standard generative

networks is that the output of the generator is limited to data representative of the training set that it was trained on. It would, however, be more beneficial to generate ballistic data for experiments that have not yet been performed. Conditional-GANs provide an opportunity to make inter-class predictions and generate new ballistic samples that it has not explicitly been trained on.

5.1.1 Conditional Generative Adversarial Networks

The standard GAN model can capture the probability distribution of random variables from real data. They can easily learn the joint probability distribution of high-dimensional data and generate brand new samples representative of the dataset that it was trained on [Li et al., 2020]. However, a common limitation with the conventional GAN is that there is little control over the new samples that are generated, which can belong to any part of the data distribution that it was trained on rather than a specific part of the distribution that the user may wish to target. A cGAN provides an option to remedy this issue. The cGAN is similar to the conventional GAN but introduces additional information to the training process such as class labels or even data from other modalities, to allow more direct control over the generating procedure of the GAN [Mirza and Osindero, 2014]. Thus, samples can be generated by a cGAN by challenging it to generate samples specific to what it has learned regarding that particular class label.

A GAN can be extended to a cGAN if both G and D are conditioned by extra information $\boldsymbol{\lambda}$. This could be any kind of auxiliary information, such as class labels or data from other modalities [Mirza and Osindero, 2014]. In this study $\boldsymbol{\lambda}$ exists as vector of numeric class labels, but the label is powerful as it allows the algorithm to make a connection between certain distributions of data and that particular class label, the intention being that once the cGAN is trained, G can be called to generate new samples of data representative of the distribution mapped to that particular class label during training by simply passing a latent input and class label as input. A schematic diagram of the cGAN is shown in Figure 5.1. It is possible to modify the objective function of the GAN, as detailed in Section 2.8 and shown in Equation 2.28), to include the additional class labels $\boldsymbol{\lambda}$. The objective function for the cGAN can therefore be expressed as

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathrm{X}\sim\rho_{data}(\mathrm{X})}[\log D(\mathbf{X}\,|\,\boldsymbol{\lambda})] + \mathbb{E}_{\mathrm{Z}\sim\rho_{\mathrm{Z}}(\mathrm{Z})}[\log(1 - D(G(\mathbf{Z}\,|\,\boldsymbol{\lambda})))] \tag{5.1}$$

Training sets

The Lambert and Jonas relation, as detailed in Section 1.5, was used to prepare the training sets in this study. This was selected for two reasons: (i) the relation can be used to create the ballistic curves for each class to form the training sets to effectively test proof-of-concept,

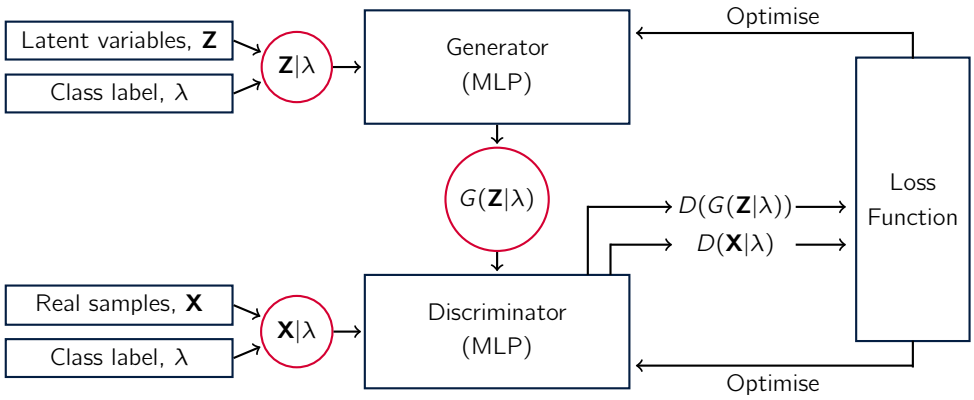


Figure 5.1: Schematic diagram of a Conditional Generative Adversarial Network (cGAN).

and (ii) it provides a useful metric through which to directly compare the output of the cGAN as the corresponding "true" values can be obtained analytically. For all data attained by the Lambert equation in this study, a and p are assigned constant values of 1 and 3 respectively. Ten different classes are considered in this study, where each class corresponds to a ballistic curve with different ballistic limit velocities, ranging from 50 to 500 m/s at 50 m/s increments. The influence that the number of samples within each class affects the training process and the output of G is then investigated. Five different cGAN models are trained, with 5, 10, 15, 20 and 25 samples within each of the 10 classes. An example training set with 20 samples within each class is shown in Figure 5.2. Due to conservation of energy, only values where $v_r \leq v_i$ are considered.

Model architecture

Both G and D exist as MLP networks as they are well equipped to deal with regression tasks and the adversarial modelling framework is straightforward to apply when both models are MLPs [Goodfellow et al., 2014]. The discriminator model D takes an instance from either the generator or the training set, along with the respective class label as input, and outputs a classification prediction as to whether the sample is real or fake. It is a binary classification problem and thus during training the model minimises the binary cross entropy loss function presented in Section 2.4.2.

The Adam version of the stochastic gradient descent method [Kingma and Ba, 2014] was selected to optimise the model parameters during training. The hyperparameters of the model were found through a grid search and the optimiser was initialised with a learning rate of 0.01 and the exponential decay rates for moment estimates β_1 and β_2 were assigned values of 0.9 and 0.99 respectively. A LReLU activation function was selected to moderate the output

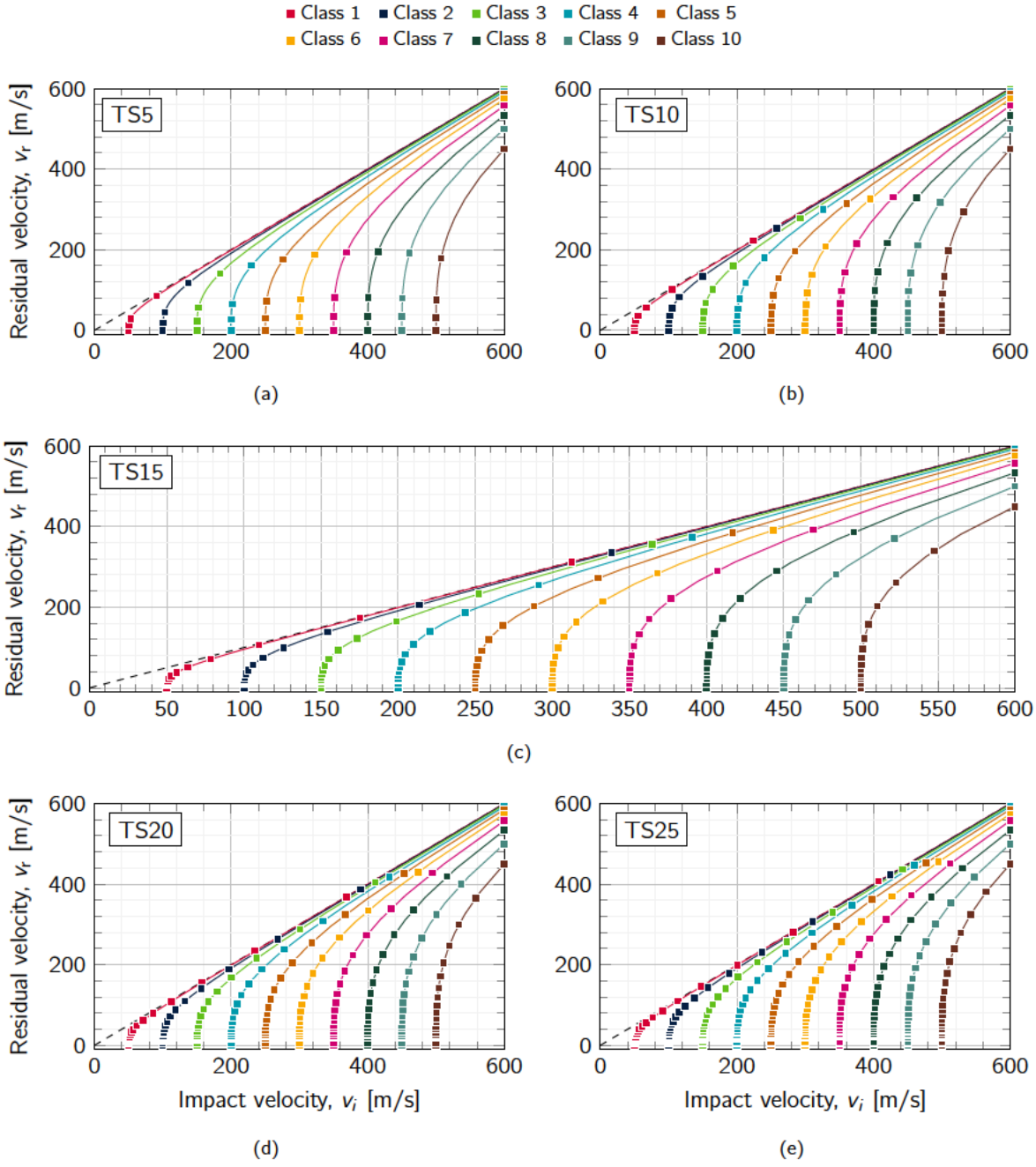


Figure 5.2: Plots (a), (b), (c), (d) and (e) show the training sets used to train cGAN models TS5, TS10, TS15, TS20 and TS25, respectively. Training sets formulated using the Lambert and Jonas ballistic model [Sikarwar et al., 2014]. The plot features 10 classes with integer labels from 0-9 that correspond to ballistic limit velocities ranging from 50-500 m/s.

from each of the hidden layers in the D model [Xu et al., 2015]. The LReLU moderates the output by allowing positive inputs to pass through unchanged such that $f(x) = x$ for $x > 0$. For negative inputs LReLU allows a shallow non-zero negative gradient. The value of the non-zero gradient is governed by parameter α as detailed in Section 2.3.3. This is contrary to the typical ReLU activation function where for negative input values the output is zero such that $f(x) = \max\{0, x\}$ [Maas et al., 2013]. Finally, the output layer of the D model passes through a Sigmoid activation function to moderate the output values in the range $[0, 1]$ [Han and Moraga, 1995].

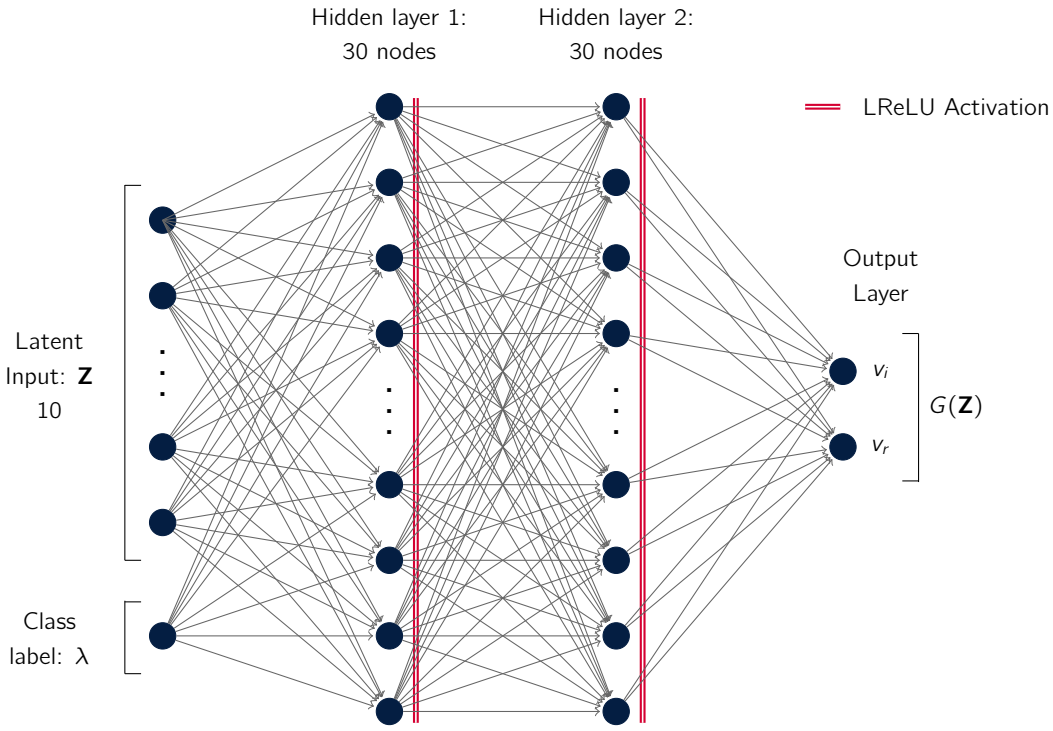


Figure 5.3: Schematic diagram of cGAN Generator.

The generator model G takes an input $\mathbf{Z}$ from the latent space along with a class label respective to the desired class of output. Once trained, it is possible to assign a class label to the G input to allow for more localised control of its output. A latent variable is a hidden or unobserved variable, and the latent space is a multi-dimensional vector of these variables. Model G simply uses 10 latent variables in its latent space that exists as a 10-element vector of Gaussian random numbers. The G network has 2 fully-connected hidden layers each containing 30 active nodes, the outputs of which are moderated by passing through a LReLU activation function with $\alpha = 0.2$. The weights associated with each node are initialised with uniform scaling between 0.05 and 1.5. The output layer exists as a vector of 2 elements, where the first

element corresponds to the predicted impact velocity and the second element to the predicted residual velocity. The output layer of the G network is consistent with the input layer to the D network. D takes input from fake samples generated by the G network and from real samples directly from the training set. These samples then pass through 4 fully connected layers that constitute the hidden layer of the D network. The layers contain 25, 15, 10 and 5 nodes respectively, the weights of which are initialised with the Glorot Uniform Initialiser [Li et al., 2020]. Once again, the outputs of each of the fully connected layers are moderated by passing through a LReLU activation function with $\alpha = 0.2$.

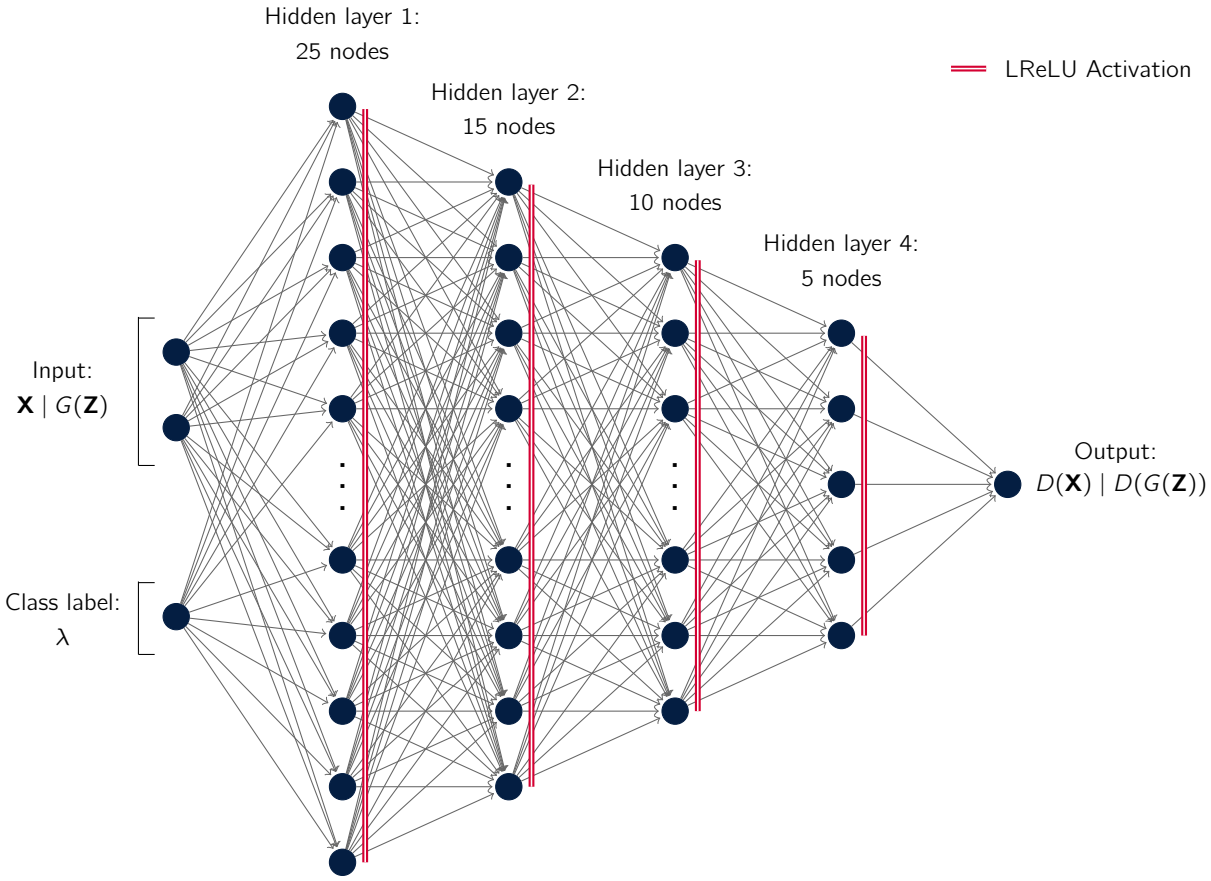


Figure 5.4: Schematic diagram of cGAN Discriminator.

Training algorithm for Initial cGAN model

The machine learning algorithm was written in Python3, using TensorFlow's high-level Keras API for building and training deep models. Keras' functional API was used as it provides

additional flexibility when creating models as it can handle models with non-linear topology, shared layers and multiple inputs or outputs. Support for multiple inputs makes it possible to also include the class labels as input to both the D and G networks that otherwise wouldn't have been supported with a traditional Sequential model. The class labels λ, that augment the training set in this study are presented to networks D and G as single integer values between 0 and 9. The performance of the cGAN model is also evaluated on non-integer class labels between 0 and 9 to consider its performance on unseen classes. In addition, the model is also evaluated on additional unseen class labels between 9.5 and 20 in increments of 0.5 The class labels λ in this study are presented to networks D and G were not encoded into one-hot vectors as first shown by [Mirza and Osindero, 2014]. A one-hot vector is a representation of categorical variables as binary vectors; each categorical variable is represented as a vector of zeros except the index of that integer that is marked as 1. In this study, the capabilities of cGAN networks are tested on class labels that are not present within the training set and by one hot-encoding the class labels, this would not be possible. This method allows non-integer class labels to be input to the network to evaluate a continuous response of how the output of the cGAN varies depending on the class label given.

The stopping criteria for training the cGAN is met once the model has been trained for 50,000 iterations and a single iteration is complete once the parameters of both the D and G networks have been updated. Within each iteration, the trainable parameters of both the D and G networks are updated alternatively: first, the D network is trained and then the G network. The discriminator D is simply a classifier that over time learns to distinguish real data from the training set (label = 1) from fake data generated by G (label = 0). During training, the trainable parameters of the network are updated accordingly via the binary cross entropy loss function and the adam optimiser such that it can distinguish between real and fake samples when making classifications. D can therefore be decoupled and trained independently as it is trained on both data from the training set and from fake data generated by G. At the start of each iteration, a number of real samples (with target label = 1) and associated class label $(0-9)$ are randomly selected from the training set $\mathbf{X}$ and used to train D. G in its current form is then used to generate the same number of fake samples (with target label = 0) and D is trained again. The number of samples passed through the network for training in one iteration is dictated by the batch size. It is important to note that during each iteration the parameters of the D network update twice as often as the G network. This is simply because it is trained on both real and fake samples. If left uncorrected, D would train at a faster rate than G as it is exposed to twice as much data, thus giving D an advantage and subsequently affecting the competitive nature of the adversarial process. In order to correct this, the batch size used to train the discriminator is halved such that the combined total number of samples from the training set and that generated by the generator is the same as the number of samples used to train G.

The G network is trained entirely on the performance of D and its trainable parameters

are optimised in accordance with its output $D(G(\mathbf{Z}))$. When D is successfully classifying fake samples G is updated more rigorously. Conversely, when D is less successful at classifying fake samples then D is updated more rigorously. This defines the zero-sum adversarial relationship between the two models. As the training of G depends on the output of D, the network cannot be decoupled and trained in isolation. A combined model is required to form the GAN network. The GAN network is a sequential model that stacks both the G and D networks such that G receives the latent Gaussian vector as input and can directly feed its output into D. The output of this larger model $D(G(\mathbf{Z}))$ can then be used to update the trainable parameters of G by once again using the binary cross entropy loss function and adam optimiser with a learning rate of 0.002 and β_1 and β_2 were given values of 0.9 and 0.99 respectively. Within the same training iteration, the G network is used to generate a number of samples equivalent to the batch size. It should be emphasised here that fake samples are passed through the combined GAN model with a label of 1 indicating that the samples are real — this is because G wants to optimise its parameters such that D will classify fake samples as real and this step is a key part of that training process. It should be noted that whilst G is being trained within the combined model, the nodes within each layer of the D network are frozen and cannot be updated. This prevents D from being trained on fake data with incorrect labels. The training algorithm of the entire process is shown in Table 5.1.

5.1.2 Model evaluation

A total of 5 cGAN models have been trained for a total of 50,000 iterations. Each model was trained on a multi-class training set containing 10 classes of ballistic data labelled with an integer from 1 to 10, where each class represents a ballistic curve with a different ballistic limit velocity ranging from 50 to 500 m/s at 50 m/s increments. Five training sets (TS) were prepared and labelled as TS5, TS10, TS15, TS20 and TS25, where the number refers to the number of ballistic samples present in each class. In this study, the cGAN models are evaluated in two ways:

1. To generate additional samples that belong to each of the 10 classes that it was trained on.
2. To generate additional samples that belong to classes that it was not trained on, i.e. intermediate ballistic limit velocities and class labels that exist beyond the domain of the training set.

Each class refers to a ballistic curve with a particular v_{bl}; for example, class labels 0, 1 and 2 refer to a ballistic curve with v_{bl} equal to 50, 100 and 150 m/s, respectively. It is therefore possible to generate an expected ballistic curve for any class label using the Lambert model and

Table 5.1: Training algorithm for the cGAN.

Inputs: Training Set $\boldsymbol{X}$, latent dim $\boldsymbol{Z}$, class labels $\boldsymbol{\lambda}$, iterations T, batch size m
Outputs: Generated samples $G(\mathbf{Z})$, $D(\mathbf{X})$, $D(G(\mathbf{Z}))$
Context: A sample from X from $\boldsymbol{X}$ is a 2 element vector that relates to a single point on the ballistic curve, where v_i and the corresponding v_r refer to the x and y coordinates respectively.

start
Define: D and G MLP networks
Define: Combined sequential cGAN model with D and G
Initialise: Trainable parameters $\boldsymbol{\theta}$ of D and G
Freeze: Trainable parameters of D in combined cGAN model

for t = 1 : T **do**
Collect samples $\{X_i\}_{i=1}^{m/2}$ from training set $\mathbf{X}$, label = 1
Generate samples $\{G(\mathbf{Z}, \lambda)\}_{i=1}^{m/2}$ from distribution $P_{(\mathrm{Z}\,|\,\lambda)}$, label = 0
Train D on real samples $\{\boldsymbol{X}_i\}_{i=1}^{m/2}$ and update D, label = 1
Train D on fake samples $\{G(\mathbf{Z}, \lambda)\}_{i=1}^{m/2}$ and update D, label = 0
Invert label of $\{G(\mathbf{Z}, \lambda)\}_{i=1}^{m}$ from 0 to 1
Train GAN on $(\mathbf{Z}, \lambda)\}_{i=1}^{m}$ with inverted labels and update G
end for
end

the samples generated by the cGAN networks can be compared directly with those expected values. To that end, a method is established to compare and evaluate the predictions of the cGAN directly and test the quality of the approach.

Inspection of cGAN model output

When provided with a latent input of the correct size and a class label, the trained G model outputs a 2 element vector, or sample, that contains the impact velocity v_i and the corresponding residual velocity v_r. The model can then be used to generate samples and plot them against the expected ballistic curve. Plotting the results in this way is useful to get a visual understanding of the model's output and to identify any problem areas where the output may struggle or fail to map a particular feature of the training set. It should be noted that the output of the cGAN network is stochastic and if a model is used to generate 100 samples twice, it is unlikely that the 100 samples from each case are identical. The samples from each case however, belong to the same probability distribution that the cGAN model has learned as a result of

the training process. The evaluation of the models therefore begins by simply using each of the cGAN models to generate 200 samples for each of the class labels within the training set (labels $0-9$). In this regard, if the model is successful it would be expected that its output is similar to that of the training set and it would be possible to identify 10 distinct ballistic curves with the correct v_{bl}. Whilst inspection is a useful tool for assessing the output of generative networks, it is important to perform a more thorough statistical analysis for comparison. This analysis is split into two main points of evaluation:

- How accurately can the cGAN models predict the ballistic limit velocity?
- How close do the generated samples match the expected ballistic curve?

Ballistic limit velocity predictions

A non-linear least squares method is used to fit the generated samples from each model on each class to the Lambert model's parameters a, p and v_{bl}. Each cGAN model was used to generate 10,000 samples belonging to each class as this was deemed sufficient to capture the entire distribution of the cGAN networks. The key parameter of interest is v_{bl}; researchers perform ballistic experiments with the primary intention of determining this parameter as it plays a crucial role in the design and development of armour plates and other protection systems. In order for generative networks to have a place within the space of ballistic testing and design, it is necessary that they can be used to provide useful v_{bl} predictions. The benefit of this study is that the expected v_{bl} for each class are known and thus the predicted v_{bl} from each of the cGANs can be compared directly with the expected value, and the difference can be calculated to determine the prediction error. The study first uses this method to predict the v_{bl} on integer class labels that have been seen (i.e. integer class labels $0-9$) before analysing the performance of the cGAN on classes that were not present in the training set, such as non-integer class labels in $[0, 9]$ and also class labels that exist beyond the domain of the training set such as labels in $[9.5, 20]$.

Assessing the variability of generated samples

Whilst it is necessary that the cGAN models can be used to predict the v_{bl}, this does not give any indication to the variability or the spread of the data. It is possible that the fitted v_{bl} predictions could be the same for two different models where samples generated from one are more spread than from the other.

For this study, a favourable model is one that can accurately predict the v_{bl} for a particular class, whilst also being capable of generating samples that lie close to the respective ballistic

curve. One way to assess the spread of the data is to use the Lambert equation to calculate the equivalent v_r values for each v_i generated by the cGAN and determine the RMSE between the predicted and expected values. However, just after v_{bl} the ballistic curve has a steep gradient and slight variations in v_i would return vastly different v_r values. This becomes problematic as large discrepancies may be returned for samples despite lying close to the ballistic curve. It is therefore important to define what a "perfect" generated sample looks like. A single sample consists of a 2 element vector containing a value for both v_i and v_r, and a perfect sample is one where those generated values lie exactly on the expected ballistic curve, as predicted by the Lambert model. There are therefore many locations on the Lambert curve that could be described as "perfect", where the respective error would be 0%. In this study, the generated samples produced by each of the models are compared with that of the nearest point on the ballistic curve, and the performance is evaluated in three ways. First, the distance d from each of the generated samples to the nearest point on the ballistic curve is calculated as

$$d = \sqrt{(x_g - x_l)^2 + (y_g - y_l)^2} \tag{5.2}$$

where x_g and y_g are the respective impact velocity and residual velocity of the generated samples, and x_l and y_l are the coordinates of the Lambert curve closest to the generated sample as shown in Figure 5.5. In this case d returns the distance as an absolute value. This is obtained for each cGAN model for each of the 10 classes. The RMSE is then calculated with respect to both the x values of the generated samples (v_i) and the y values (v_r) for each model on each class. It is therefore possible to obtain a vector containing the minimum distances d from each sample to the ballistic curve and evaluate the distribution .

The KDE is used to estimate the probability density function of the variable d and present the shape of the distribution. In this assessment, the distances from 2,000 generated samples to the expected Lambert curve were computed for each cGAN model for integer classes from $0-9$. The associated d values from all classes are combined into a single vector of residuals for each cGAN model to assess its performance. It should be noted that the polarity of d is included by comparing the coordinates of the generated sample to those of the nearest neighbour on the corresponding ballistic curve. A KDE assessment is first performed on seen classes with labels $0-9$ before comparing the distributions from unseen class labels. The evaluation on unseen class labels is split into two parts: for non-integer class labels that exist within the domain of the training set (0.5, 1.5, . . . , 8.5), and for unseen class labels that exist beyond the scope of the original training set (9.5, 10, . . . , 19.5, 20).

5.2 Results and Discussion

This section begins by considering the first of the evaluation criteria where each of the cGAN networks were used to generate samples that belong to each of the 10 classes it was trained

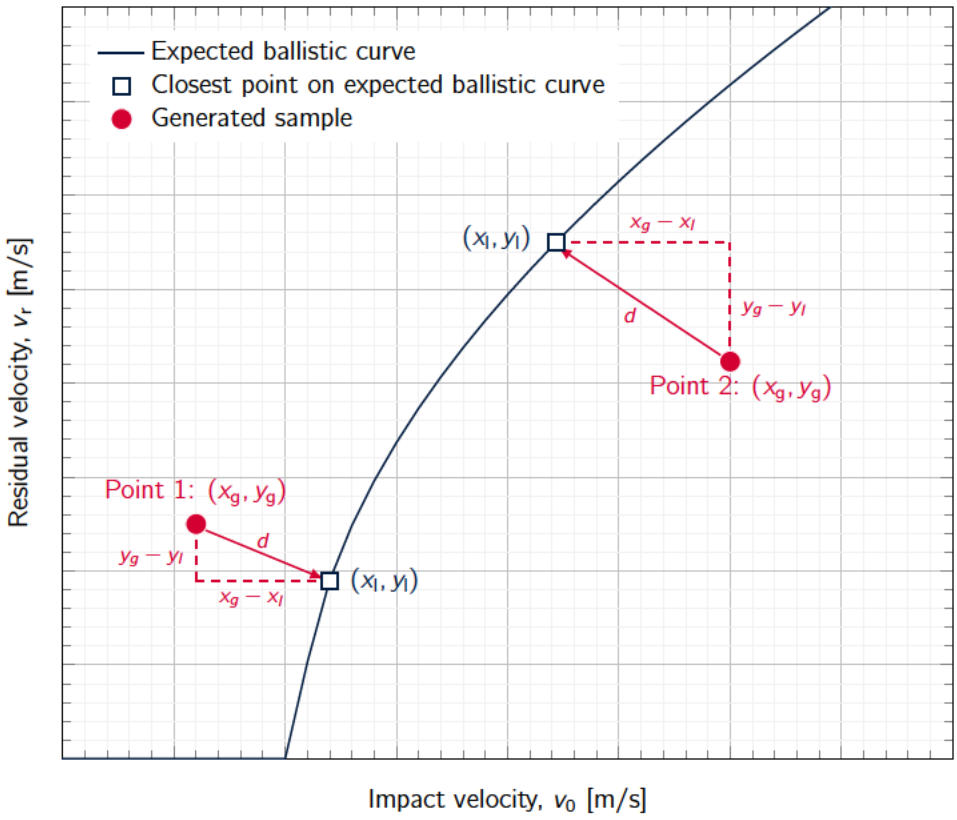


Figure 5.5: Graph indicating the minimum distance between generated samples and the closest point on the expected ballistic curve.

on. Figure 5.6 shows an example output from each of the 5 trained cGAN models (TS5, TS10, TS15, TS20 and TS25). Each model was used to generate 100 samples from each of the 10 classes, the results from each model are presented in Figure 5.6a to 5.6e, respectively. It can be seen that the TS5 GAN was the worst performing model and was also trained on the training set with the lowest number of samples for each class, 5 in this case. Whilst the model is able to learn the distinction between classes, the samples it generates do not align well with that of the Lambert curve. At low v_r values the generated samples do not follow the ballistic curve and for higher class labels the TS5 cGAN fails to generate samples at $v_r = 0$. For each class, the samples generated at the v_{bl} are similar to the location of the v_{bl} in its training set however, the samples are dispersed indicating uncertainty in the model. The TS5 prediction on class 1 (TS5: 1) is the closest to its respective Lambert curve with its prediction on class 10 (TS5: 10) being the worst. An interesting observation with the multi-class training set is that for higher ballistic limit velocities the curvature of the ballistic curve is exaggerated and becomes more non-linear. This suggests that the TS5 cGAN struggles to map this non-linearity as for higher residual velocities the generated samples appear further away from the corresponding

Lambert curve. It can also be seen that the TS5 model generates some samples from classes 1, 2 and 3 that exist beyond the $y = x$ boundary, as shown in Figure 5.6. For these samples, $v_r > v_i$, which violates conservation of energy through an increase of kinetic energy of the projectile. Generated samples that satisfy the condition $v_r > v_i$ could be omitted in practice, but they were included in this study to evaluate the accuracy of the cGAN output.

The output from cGAN models TS10, TS15, TS20 and TS25 was more successful. In each case, the cGAN models have successfully generated additional samples that follow the shape of the Lambert ballistic curve. The generated samples appear to come from the same distribution as the training set and closely match the Lambert model, both at the v_{bl} and for higher residual velocities for each class. This is a clear indication that the cGAN model has learned directly from the training set, but more importantly that it has demonstrated the capability of the model to produce class-specific outputs, corresponding to the input label it was given. This is particularly powerful as it addresses the issue of limited control regarding the output of the model of conventional GANs. When comparing Figures 5.6b to 5.6e with the output from the TS5 cGAN in Figure 5.6a, it can be seen that the samples generated along the length of the ballistic curve show less variation than the output of the TS5 cGAN model. This could be an early indication that 5 samples for each class in the training set is insufficient for this particular cGAN architecture to effectively match the distribution of the data. Also, the TS5 cGAN model in particular demonstrates a bias to generate samples with a very low v_r as of the 100 samples generated, the majority lie in this region.

5.2.1 Non-linear least squares ballistic limit velocity predictions

The non-linear least squares method was used to fit the generated samples from each model on each class to parameters from the Lambert model. The non-linear least squares method is a popular approach to fitting data to a model function that is non-linear in its parameters. It is well suited in this case as the Lambert model consists of 3 parameters a, p and v_{bl} that can be determined using this method. The optimisation algorithm attempts to find the values of the unknown parameters that minimise the objective function. The optimisation algorithm used for this is the Levenberg-Marquardt algorithm. The model evaluates the goodness of fit by examining the differences between the actual data and the model predictions, the residuals, and computing various statistical measures such as the R-squared coefficient. Each cGAN model was used to generate 10,000 samples belonging to each class and the v_{bl} was obtained. Table 5.2 contains each of the models' v_{bl} predictions and their associated percentage error. Figure 5.7 shows the absolute percentage difference of the fitted v_{bl} predictions with the expected values. It can be seen that in all cases, the $|v_{bl}|$ percentage error was less than 10%. It can also be seen that the TS5 cGAN model was the worst performing as it returned the highest error for each class label other than class 2. TS5 performed particularly poorly for higher class labels when

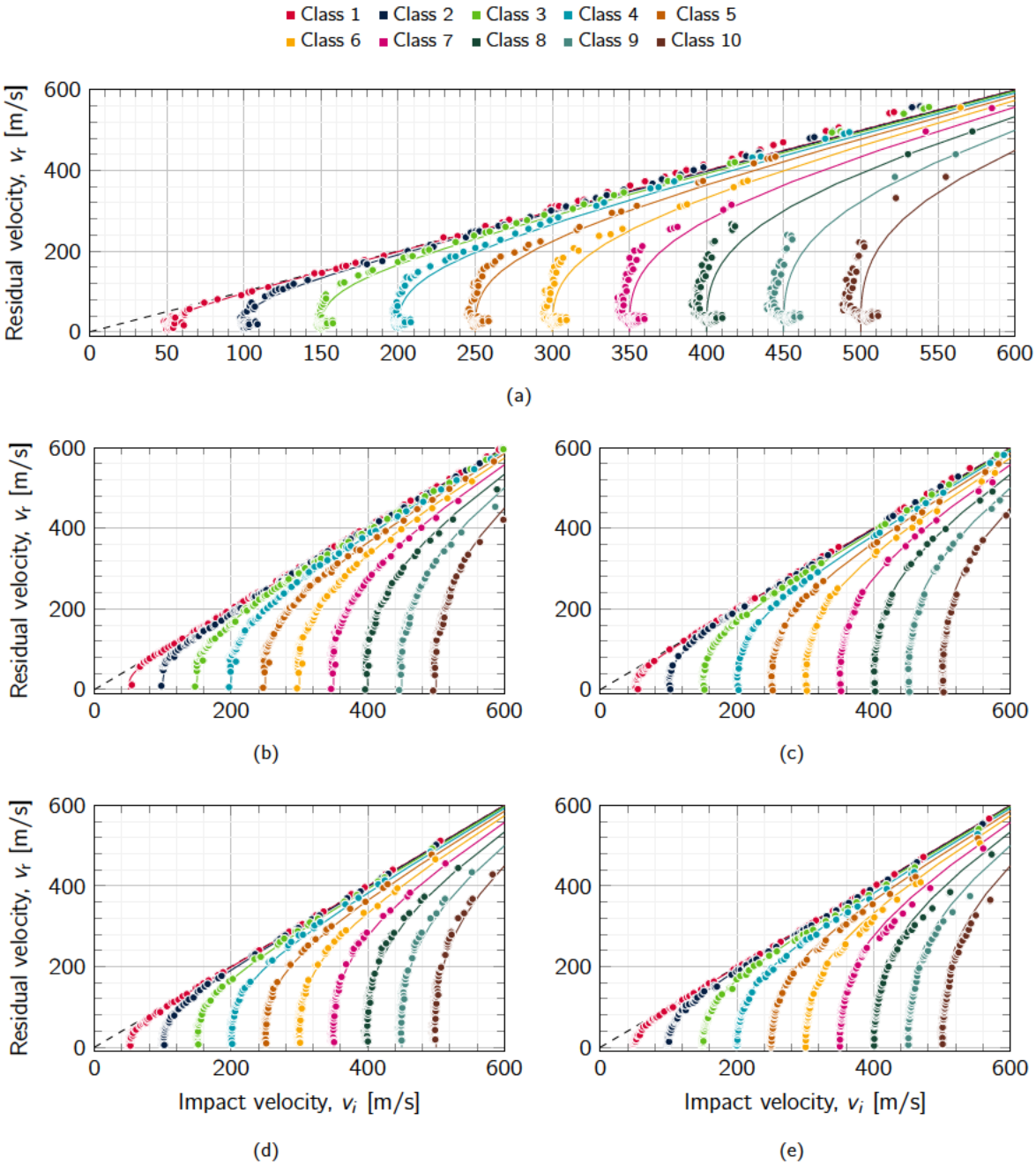


Figure 5.6: cGAN model output from (a) TS5, (b) TS10, (c) TS15, (d) TS20 and (e) TS25 cGANs. The cGAN networks were used to generate 100 new samples for each of the 10 classes.

compared to the other models. For class label 9, TS5 returned a $|v_{bl}|$ error of 9.2% whereas the next highest was TS10 with a value of 1.1%. For higher class labels the $|v_{bl}|$ of TS5's predictions continues to increase.

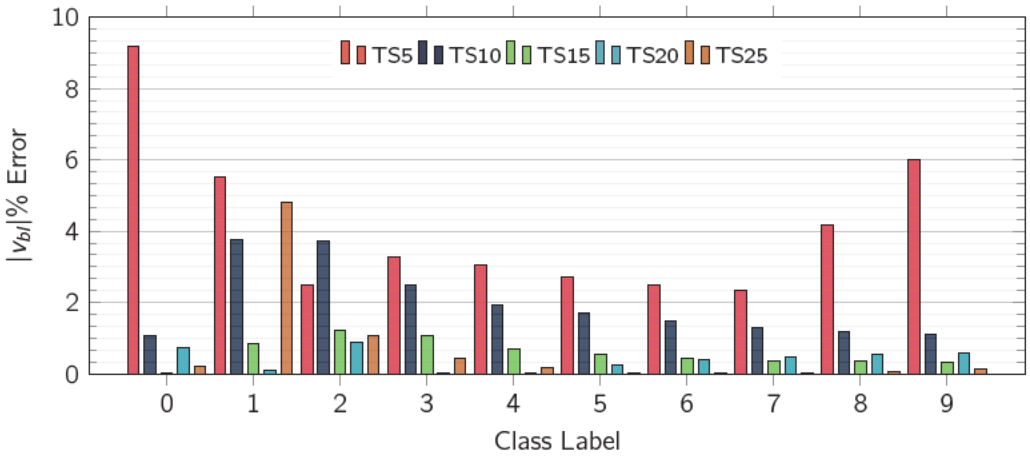


Figure 5.7: Relative error of v_{bl} predictions for cGAN models TS5, TS10, TS15, TS20 and TS25 when compared to that of the true v_{bl} vales from the Lambert model. Predictions determined via the non-linear least squares method to fit 10,000 generated samples from each model on each class to parameters a, p and v_{bl} from the Lambert model.

cGAN models TS15, TS20 and TS25 were the best performing ones. For class labels $1-9$, the predictions of v_{bl} were made with an error of less than 1.2% to the Lambert model. Interestingly, the TS25 model trained on the largest training set performed relatively poorly on class label 0 with a $|v_{bl}|$ error of 4.8%. The average errors for each of the cGAN models across each of the 10 classes are listed in Table 5.2. It can be seen that TS5 was the worst performing model with v_{bl} predictions with an average error of 4.1% and TS10, TS15, TS20 and TS25 with average errors of 2.0, 0.6, 0.3 and 0.7%, respectively. It should be noted that these values were obtained using the non-linear least squares method by fitting the generated samples to the parameters a, p and v_{bl} from the Lambert model. For TS5 in particular, whilst it appears to determine the v_{bl} with an average error of 4.1%, it is not necessarily indicative of a model that can generate new samples accurately. For higher class labels TS5 was incapable of generating samples at the v_{bl}, as can be seen in Figure 5.6. The remaining cGAN models generate samples that follow the ballistic curve and generate samples at the v_{bl} with little dispersion. This means that predictions for the v_{bl} determined by the fitting method are better suited to these models.

Table 5.2: Ballistic Limit Velocity, v_{bl}, predictions of trained cGAN models TS5, TS10, TS15, TS20 and TS25. Table shows v_{bl} prediction for each model for the 10 classes and the percentage error when compared to the true Lambert model. Predictions determined by using the non-linear least squares method to fit 10,000 generated samples from each class to parameters a, p and v_{bl} from the Lambert model. The table also presents the average absolute error for each cGAN model across each of the 10 classes.

Class	TS5	TS10	TS15	TS20	TS25
0	47.3 (−5.5%)	51.9 (+3.8%)	49.6 (−0.9%)	50.0 (+0.1%)	47.6 (−4.8%)
1	97.5 (−2.5%)	96.3 (−3.7%)	101.2 (+1.2%)	101.0 (+0.1%)	98.9 (−1.1%)
2	154.9 (+3.3%)	146.3 (−2.5%)	151.2 (+1.1%)	150.0 (+0.0%)	149.3 (−0.5%)
3	206.1 (+3.0%)	196.1 (−1.9%)	201.4 (+0.7%)	199.9 (−0.0%)	199.6 (−0.2%)
4	256.8 (+2.7%)	245.8 (−1.7%)	251.4 (+0.6%)	249.4 (−0.3%)	250.1 (+0.0%)
5	307.5 (+2.5%)	295.5 (−1.5%)	301.3 (+0.4%)	298.9 (−0.4%)	300.1 (+0.0%)
6	358.2 (+2.3%)	345.4 (−1.3%)	351.3 (+0.4%)	348.3 (−0.5%)	350.1 (+0.0%)
7	383.3 (−4.2%)	395.2 (−1.2%)	401.4 (+0.4%)	397.7 (−0.6%)	399.7 (−0.1%)
8	423.0 (−6.0%)	444.9 (−1.1%)	451.5 (+0.3%)	447.4 (−0.6%)	449.4 (−0.1%)
9	454.1 (−9.2%)	494.7 (−1.1%)	499.9 (−0.0%)	496.3 (−0.7%)	499.0 (−0.2%)
Average	4.1%	2.0%	0.6%	0.3%	0.7%

5.2.2 Model variation analysis and distributions

The average minimum distance between the samples generated by the trained cGAN models and their respective ballistic curve for each of the 10 classes is shown in Figure 5.8a. It should be noted that the distance metric in this case, refers to the hypotenuse between the difference in the x coordinate (v_i) and the y coordinate (v_r) between the generated and expected samples and is presented as an absolute value. The "distance" between points is therefore determined with unit of m/s and is presented as a means for comparison rather than a relative value. In this analysis, the cGAN models were used to generate 2,000 samples for each class and the absolute minimum distance d was determined for each, and the average $\bar{d}$ taken. It can be seen that on average samples generated by the TS5 cGAN network are further away from the expected ballistic curve than the remaining models, that is, the samples generated by the TS5 cGAN network are more spread out and present more variation. This was consistent for each of the 10 classes.

The output of the remaining cGAN models (TS10 to TS25) was consistently lower than the TS5 cGAN and values of $\bar{d}$ were comparable to one another, indicating that the samples generated by these networks are less spread out and lie closer to the expected values. Figure 5.8b presents the RMSE error when comparing the generated x values (v_i) with the expected x values from the Lambert model. As expected, TS5 was the worst performing model with errors ranging from 6.73 to 14.91% across class labels $0-9$. The remaining models returned errors of less than 5.80% across all classes. The TS15 model was the best performing model with an

RMSE error of less than 2.39% for all classes. Finally, Figure 5.8c shows the RMSE error with respect to the y coordinate that represents the residual velocity v_r. Similar to the results in Figure 5.8b, the RMSE errors for models TS10 to TS25 were all below 5%. Again, model TS5 was more prone to errors returning average errors above 10% for class labels $0-3$ peaking at 19.02%. It should be noted that for class labels $6-9$ the RMSE errors of TS5 were consistent with those of the remaining models. However, the distance from the generated sample to the expected value on the ballistic curve is a product of the errors in both the x and y coordinate for v_i and v_r, respectively. It is clear that in this case TS5 is the worst performing model.

To effectively compare the remaining cGAN models, Figure 5.9 shows the Kernel Density Estimation (KDE) of the probability density function of the variable d. The distances from 2,000 generated samples to the expected Lambert curve were computed for each cGAN model for integer classes $0-9$. The KDE distribution for the TS5 cGAN is a bi-modal, left-skewed negative distribution with peaks at $d = 1.14$ and $d = -7.64$. The distribution also appears flatter than the remaining cGAN models, indicating that the distance d of samples generated by this model to the expected ballistic curve are more spread out and varied. This results in a wider range of possible distances to the expected values, and subsequently a less accurate model. The TS5 distribution is consistent with that of the output in Figure 5.6a as the samples generated by the model demonstrate the tendency to under-predict v_i. The distributions of the remaining cGAN models are uni-modal and have a single peak indicating that the data count at that area is higher than anywhere else on the graph. For each of these models, the peak lies close to $d = 0$ indicating that the majority of the samples produced by the models are close to the ballistic curve. The distributions of models TS10, TS15 and TS20 appear symmetrical and could be classified as normal distributions, whereas TS25 is a right-skewed positive distribution. Models TS10 and TS20 indicate a slight tendency to generate samples with a lower v_i as the peaks of these distributions exist where d is negative. Conversely, models TS15 and TS25 demonstrate the opposite and with their peaks existing at positive values of d. It should be noted that the peaks for distributions TS15 and TS25 are at $d = 0.41$ and 0.30, respectively, and thus the majority of samples generated by these models lie close to the expected ballistic curve. Overall, the KDE plot shows that the TS15 model was the most successful as it is a narrow, normal distribution with a single peak at $d = 0.34$. The narrow peak indicates relatively lower variation in the generated samples and ultimately a model that is capable of generating accurate samples for each of the 10 classes.

5.2.3 Evaluation of cGAN models on unseen class labels

This section explores the capabilities of the trained cGAN models to generate additional samples that belong to classes that it was not trained on, i.e. intermediate ballistic limit velocities. Labels are passed into the cGAN network to evaluate its response on both integer and non-integer

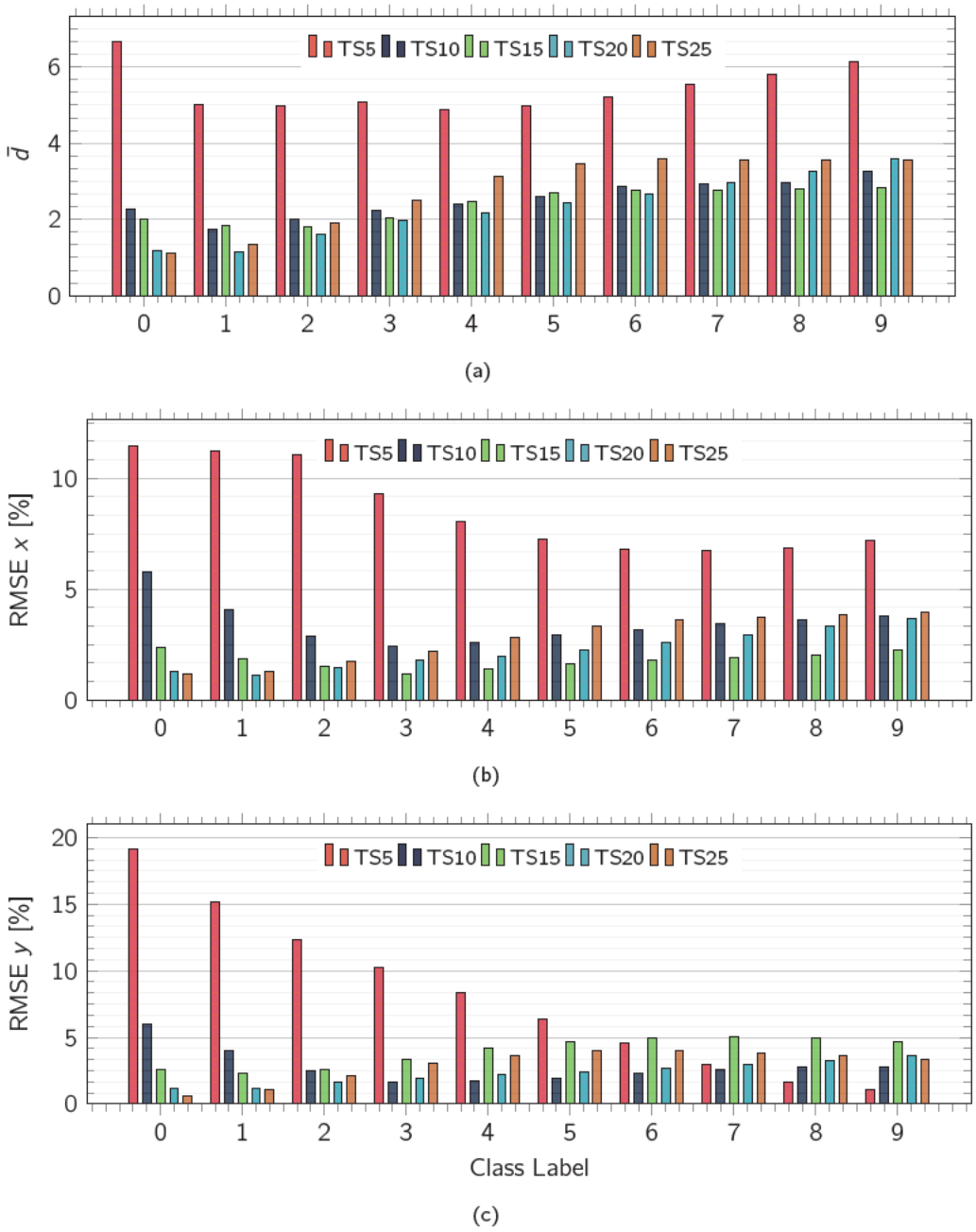


Figure 5.8: Results from cGAN networks TS5, TS10, TS15, TS20 and TS25 used to generate 2000 samples for integer class labels from 0-9. Plot (a) shows the average minimum distance from the generated sample to that of the respective ballistic curve produced by the Lambert model for each cGAN model for each class label. Plots (b) and (c) present the RMSE of the generated sample with the expected sample when comparing the x (v_i) and y (v_r) coordinate respectively.

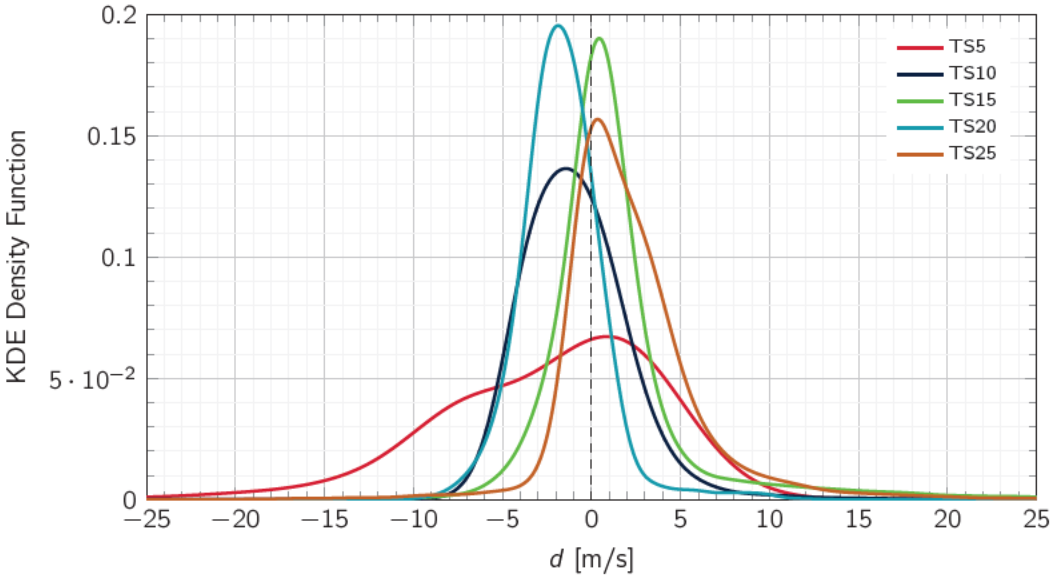


Figure 5.9: Kernel Density Estimation (KDE) of minimum distance d from the generated sample to the expected ballistic curve for cGAN models TS5, TS10, TS15, TS20 and TS25. In this instance, each model was used to generate 2,000 samples for each class and the minimum distances for all classes were combined into a single vector to form the models distribution.

class labels between 0 and 20 in increments of 0.5. The training set used to train each of the cGAN networks feature integer class labels from $0-9$, therefore non-integer class labels from $0-9$ and all labels above 9 are new to the cGAN network. Figure 5.10 shows the output from the TS15 cGAN. In this plot the model was used to generate 200 samples belonging to each of the class labels. The same latent input $\mathbf{Z}$ was provided in each case such that the variation in the output is a result of the changing class label only. It can be seen that the generated samples are consistent with the Lambert curve, even for unseen class labels. However, it is important to assess the output of each cGAN model qualitatively. Each of the models are once again used to generate samples and the fitted v_{bl} is compared with that of the expected Lambert ballistic limit velocity parameter via the non-linear least squares minimisation method on the Lambert function.

The results in Figure 5.11 compare the relative error between the v_{bl} prediction from each cGAN output with the expected v_{bl}. The predictions were obtained by fitting 1,000 samples generated by each of the cGAN networks by the non-linear least squares method and calculating the percentage difference between the obtained v_{bl} parameter with that of the Lambert model. It can be seen in Figure 5.11a that the TS5 cGAN model is the worst performing model. TS5 struggles to predict the v_{bl} for class labels outside of the domain that it was trained on. This is evident by the large spike in error for class labels between $10-20$ and the recorded a peak error

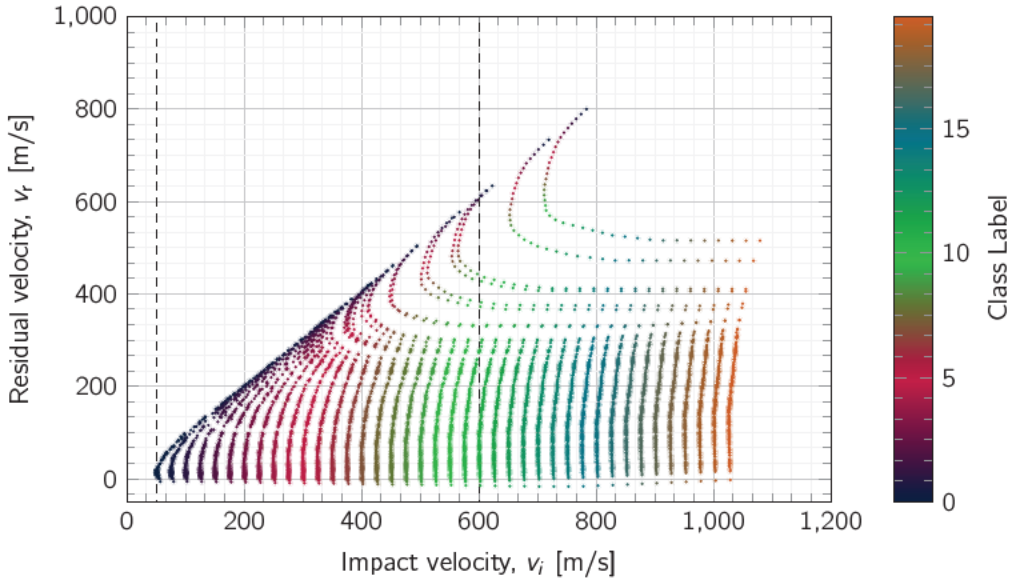


Figure 5.10: Output of the TS15 cGAN network challenged to generate 200 samples for integer and non-integer class label inputs between 0 and 20 in increments of 0.5.

of 56.7% for class label 14. Figure 5.11b is a detail view of the same results in Figure 5.11a. It can be seen that for all cGAN models in all cases the v_{bl} predictions made on non-integer class labels are consistent with those on integer class labels. This demonstrates the ability of the cGAN network to predict important ballistic parameters such as the v_{bl} for classes that it was not trained on. This is particularly evident for class labels $10-20$, as cGAN models TS10 to TS25 make accurate v_{bl} predictions consistently with a relative error below 1%.

The KDE of the minimum distance d of the generated samples to the expected ballistic curve for each of the cGAN models is shown in Figure 5.12. This section evaluates the models on unseen class labels and is split in two domains: the first domain considers non-integer class labels from 0.5 to 8.5 and the second domain considers both integer and non-integer class labels from 9.5 to 20 both increasing in increments of 0.5. Each of the cGAN models were trained on a training set that consists of class labels from $[0-9]$ and therefore the first domain under consideration belongs to non-integer samples that also exist within that boundary and the second domain belongs to those that do not, i.e. class labels that are beyond the scope of the training set. The cGAN models were used to generate 2,000 samples for each class label and the minimum distances from each sample to the expected ballistic curve were stored and combined into a single vector to form the models' distribution.

The TS5 model performed the worst for both domains of interest. For the first domain, the TS5 distribution is non-normal and appears left-skewed with a bias to generate samples that

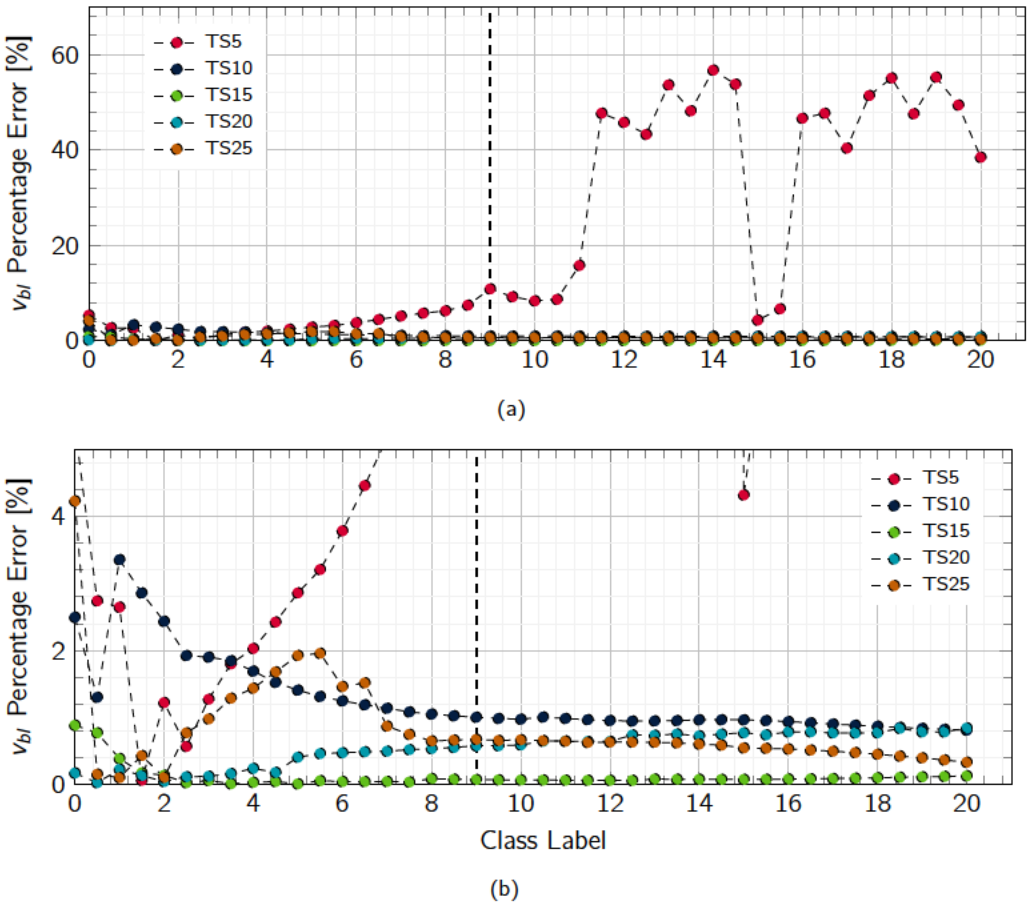


Figure 5.11: (a) Percentage error of v_{bl} predictions determined by non-linear least squares method fitted to the lambert model to the expected values. Predictions are shown for class labels 0 to 20 in increments of 0.5 for cGAN models TS5, TS10, TS15, TS20 and TS25. (b) Detail view of plot (a) containing the same results,

under-predict the expected value on the ballistic curve. The distribution appears bimodal but the peaks are shallow and not pronounced. The TS5 distribution for the second domain also indicates that the model was unsuccessful at consistently generating samples that match the expected ballistic curve. Here, the distribution is wider than that of the first domain indicating additional variability. The plot would be best described as a uniform distribution of d in the range $[-15, 10]$, with two shallow peaks at $d = -12.0$ and $d = 7.1$ m/s. The TS5 cGAN model was therefore unsuccessful at generating samples that are representative of the ballistic curve for non-integer class labels within the boundary of the training set, and for integer and non-integer samples that do not.

Model TS15 was the best performing one for both domains of interest. For the first domain,

the TS15 distribution is normal and peaks at $d = 0.27$ m/s. This is a clear indication that the cGAN model can consistently generate accurate samples for non-integer class labels from 0.5 to 8.5. It indicates that the model was able to learn the features of the training set and extrapolate its learning to class labels that it has not seen explicitly. The TS15 model also performs well on the second domain. Here the distribution appears right-skewed with a bias to generate samples that over-predict the expected value on the ballistic curve. However, the peak of the distribution remains close to 0 indicating that the majority of the samples generated by this model are still accurate. This is an excellent result given that many of the samples generated within this domain belong to class labels that are far from the boundary of the original training set, but yet the model is still able to return accurate values on samples that the model has not explicitly been trained on. This is extrapolated by comparing the distributions with the output shown in Figure 5.10 as it is clear that the model has successfully captured the characteristics of the ballistic curve and is capable of extrapolating its *knowledge* to generate accurate ballistic samples for class labels that exist beyond its training boundary.

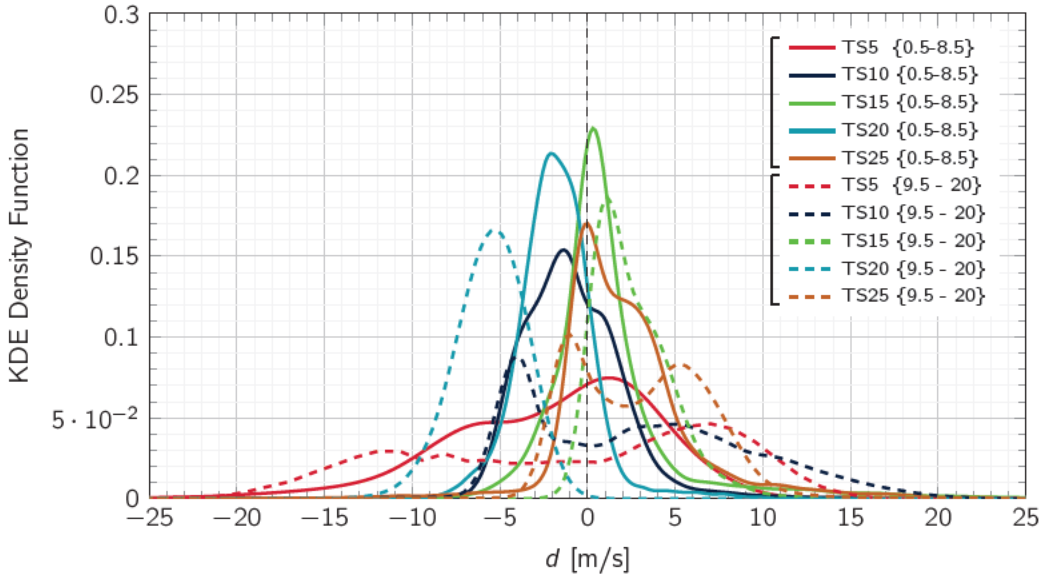


Figure 5.12: Kernel Density Estimation (KDE) of the minimum distance d from the generated sample to the expected ballistic curve for cGAN models TS5, TS10, TS15, TS20 and TS25. Distance d [m/s] is calculated as the magnitude velocity between v_i and v_r. Each model was used to generate 2,000 samples for each class and the minimum distances for each class label were combined into a single vector to form the models distribution. The plot presents distributions from two domains; the first of non-integer class labels from 0.5 to 8.5 and the second from class labels 9.5 to 20 where both increase by increments of 0.5. The first domain belongs to non-integer samples that also exist within that boundary and the second domain belongs to those that do not.

5.3 Concluding remarks

This chapter explores the use of cGAN in the space of structural design for ballistic protection. A key limitation of prototyping armor systems is the requirement for ballistic testing to be performed across different thicknesses to understand the behaviour of a material to impact. This is both expensive and time consuming due to the equipment, personnel and resource requirements needed to perform the investigations. The application of the cGAN presented in this chapter offers a novel approach to increase the speed at which ballistic predictions can be made and provides an opportunity to reduce the number of experimental testing required. The impact of which could mean that design decisions about an armor system or structure can be made earlier in the design process.

In this chapter specifically, five cGAN models were trained directly on ballistic datasets to predict the v_{bl} and generate additional ballistic samples representative of the ballistic data. This study considers a single MLP cGAN architecture that is trained on a multi-class training set consisting of 10 classes labelled $0-9$ where each class refers to a ballistic curve with a different v_{bl}. The models are labelled TS5, TS10, TS15, TS20 and TS25 and each class in the training set contains 5, 10, 15, 20 and 25 samples, respectively. For integer class labels $0-9$, on average the TS5 cGAN model predicted the v_{bl} with an error of 4.1%, the remaining models predicted v_{bl} with an error of less than 2% with the TS15 model performing best with an error of 0.6%. It was also found that the non-integer class labels from $0-9$ the v_{bl} predictions were consistent with those from class labels that the model was explicitly trained on. Moreover, each of the cGAN models were challenged to generate new samples for class labels that exist beyond the scope of the training set for labels $9.5-20$. It was found that models TS10, TS15, TS20 and TS25 were still able to predict the v_{bl} with an error of less than 1.5% in all cases.

The analysis presented in this chapter successfully demonstrates that cGAN models can be used to generate additional ballistic samples in accordance to the class label given to the generator as input. This is the case for non-integer class labels that exist within the domain of the training set that the model was trained on (labels $0-9$) and also for class labels that exist beyond the boundary of the training set (labels $9-20$). cGAN networks can be used in this way to make v_{bl} predictions and generate additional samples for a variety of multi-class ballistic datasets where each class could for example refer to a different plate thickness, or heat-treatment. The benefit of applying such AI techniques is that current modelling capabilities require physical parameter input which introduces limiting assumptions directly to the model. Depending on the complexity of the experiment, FE models can take substantial time to set up and its parameters are modified such that its output aligns with that of the experimental values. cGAN networks offer an alternative by learning directly from the ballistic data and once trained, can generate additional samples quickly.

Chapter 6

Supplementing experimental datasets using GANs

The extensive research of Børvik et al. [Børvik et al., 1999a, Børvik et al., 2001b, Børvik et al., 2003, Børvik et al., 2005, Børvik et al., 2009, Børvik et al., 1999b] centres around studies of the ballistic response of armoured steel and aluminium plates. The authors find that materials with high-strength properties are clearly linked to an optimal ballistic capacity, specifically, a higher v_{bl}. However, ultra high-strength materials come with a compromise of the strength – ductility trade-off, a long-standing dilemma in materials science [Li et al., 2016b], which leads to a tendency for brittle fracture and fragmentation under impact as shown in Figure 1.1. An example of innovation in experimental ballistic research was published by the same authors in 2020 [Kristoffersen et al., 2020b]. The study investigates AM aluminium as a candidate for ballistic protection, which would allow structural designs to profit from the flexible manufacturing advantages of using metallic AM methods.

The combination of ultra-high strength and AM, however, introduces increased complexity and costs to ballistic experimental campaigns. This was demonstrated in an experimental study on AM maraging steel by [Costas et al., 2021]. The authors report significant fragmentation for both projectile and target in the ballistic tests. Often, the target plate was rendered unsuitable for further shots, reducing the total possible number of tests and thus increasing the cost of the experiment. Despite such challenges, the application of ultra high-strength AM materials presents an opportunity for areal density reduction of ballistic protection which is a key area of interest [Vemuri and Bhat, 2011]. Therefore further work is required to optimise the strength versus ductility trade-off for AM maraging steel to improve ballistic resistance capacity.

An experimental campaign was selected in order to test the capabilities of the GAN network

presented in Chapter 4. In 2020, experimental tests on maraging steel fabricated by AM were carried out by Costas et al. [Costas et al., 2021]. The maraging steel underwent ballistic tests in both its as-printed and post heat treated states; the latter possessing yield strength of 2000 MPa (classified as ultra high-strength), almost double the former. These experimental tests are exemplary of the challenges for traditional methods in exploring such materials for ballistic protection described in this chapter. It was for this reason that the Costas et al. [Costas et al., 2021] experiments were selected as good candidates to test the application of the GAN model.

6.1 Methodology

6.1.1 Training Set

Both the as-printed and heat treated plates were impacted with full AP bullets and the steel bullet core only; the specification of which is displayed in Figure 1.3. The two material variations and the two projectile variations make a total of four individual datasets. Each dataset contains seven or eight tests with impact velocities in the range of $300 - 900$ m/s, which are listed along with respective residual bullet velocities in Table 6.1.

Table 6.1: Measured values of v_i and v_r from experimental tests by Costas et al. [Costas et al., 2021] on AM maraging steel impacted by bullet core only, and AP bullet given in m/s.

(a) As-printed; core only		(b) As-printed; full AP bullet		(c) Heat treated; core only		(d) Heat treated; full AP bullet	
v_i	v_r	v_i	v_r	v_i	v_r	v_i	v_r
376.1	0.0	354.0	0.0	420.2	0.0	416.8	0.0
412.7	201.5	407.1	139.1	532.3	0.0	478.6	159.9
416.8	210.3	443.7	218.0	549.8	0.0	513.1	0.0
424.0	223.6	526.7	234.8	610.8	0.0	536.0	62.1
466.1	269.8	569.4	384.3	646.6	510.0	742.0	561.1
640.0	494.5	837.8	747.4	713.6	442.9	822.4	653.7
922.2	802.6	933.4	854.6	747.4	574.2	926.9	819.9
				985.1	819.7		

In this study, four separate GAN models were trained on the four ballistic experimental datasets presented in Table 6.1.

6.1.2 Model Architecture

The architecture used for the GAN model is the same as that presented in Section 4.1.2. The Discriminator D is an MLP model and is presented in Figure 6.1. It's input layer consists of 2 nodes to accept a single sample of ballistic data from either $\mathbf{X}$ or $G(\mathbf{Z})$ that contains v_i and its corresponding v_r. D has 4 fully connected hidden layers with 25, 15, 10 and 5 nodes from layers 1 to 4. The outputs of each of the fully connected layers are moderated by passing through a LReLu activation function with $\alpha = 0.02$. The output layer consists of a single node reserved for either $D(\mathbf{X})$ or $D(G(\mathbf{Z}))$.

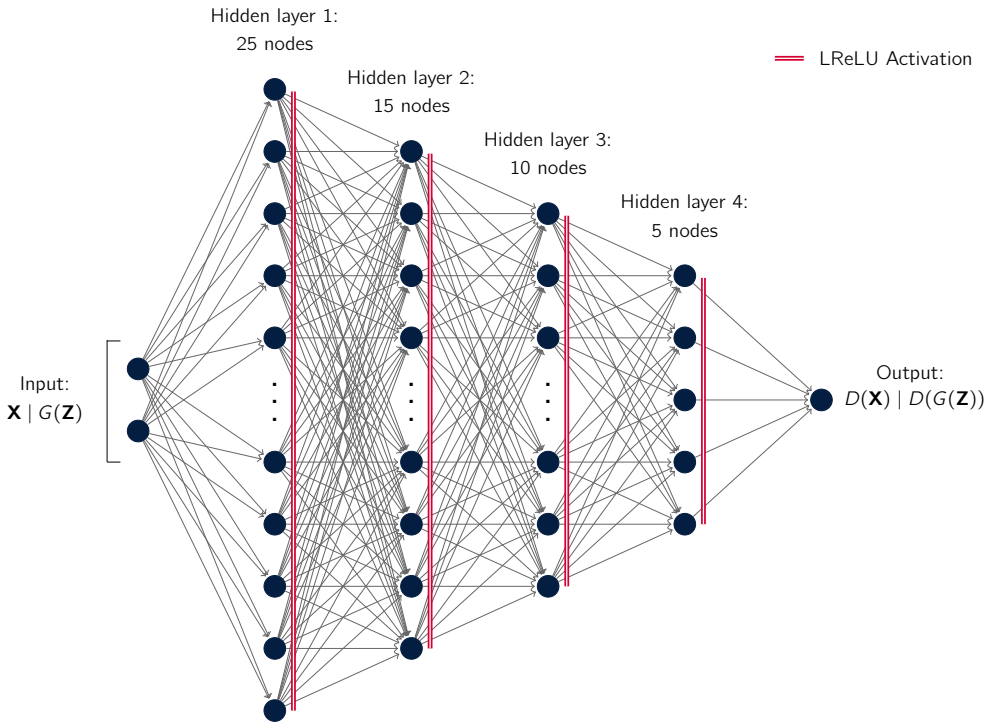


Figure 6.1: Schematic diagram of GAN Discriminator D.

The Generator G is also an MLP model and is presented in Figure 6.2. It's input layer is a fully connected layer containing 10 nodes which accepts the latent input into the network. G has two fully connected hidden layers, both of which contain 11 nodes. The outputs of each of the fully connected layers are moderated by passing through a LReLu function once again with $\alpha = 0.02$. The output layer consists of 2 nodes to represent a ballistic sample containing 2 values that correspond to v_i and v_r respectively.

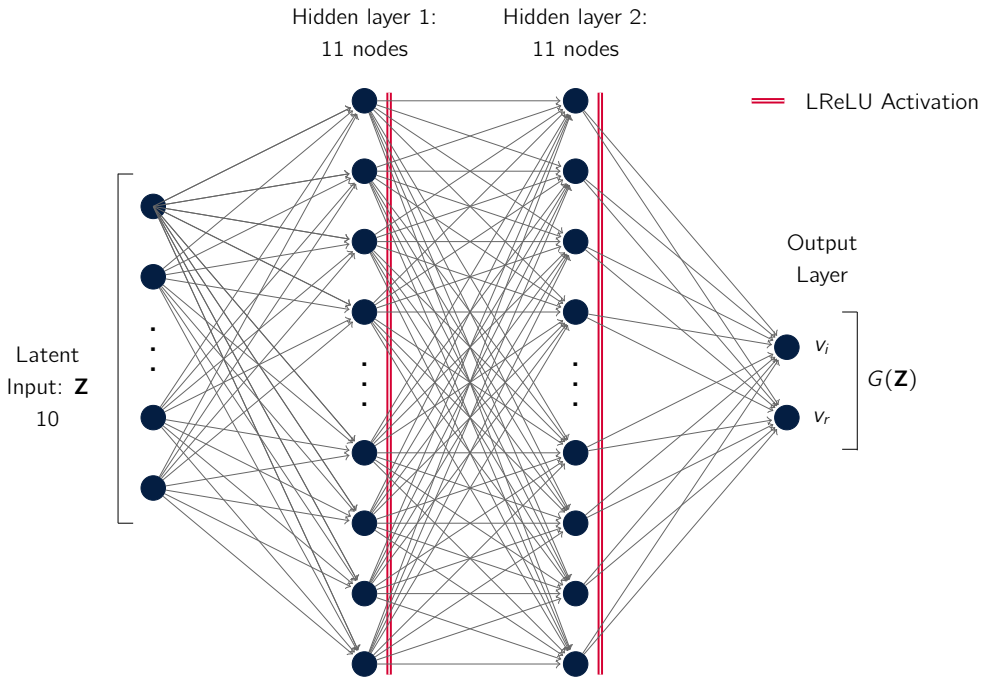


Figure 6.2: Schematic diagram of GAN Generator G

6.1.3 Training algorithm

Each GAN was trained for $1,000,000$ iterations. During each iteration, D was trained on both the *real* experimental samples and the *fake* generated samples. The model weights of D were then updated and G is trained on the classification performance of D. This process is described in greater detail in Section 4.1. The binary cross-entropy loss function was used to calculate the training loss and the adam optimiser was used with values of β_1 and $\beta 2$ of 0.9 and 0.99 respectively. D and G were also initialised with learning rates of 0.01 and 0.002 respectively.

6.2 Results

After training, G models were used independently to generate 200 *fake* samples of ballistic data. Figure 6.3 plots the 200 predictions at different stages of training for each of the experimental dataset variations (a), (b), (c) and (d). A Lambert curve was fitted to the experimental results and is plotted alongside the experimental training data. Improvement over iterations is evident in all plots, with predictions clustered closer to the training points for more developed models. No generated points after $10,000$ iterations violate energy conservation laws as they remain on

the right hand side of the $v_i = v_r$ dashed line, thus the residual velocity of the projectile does not exceed the initial impact velocity.

To estimate the GAN prediction error of v_{bl}, a Lambert curve was fitted to the generated points and compared to the experimental Lambert fit estimation of v_{bl}. The 100 point average error at $1,000,000$ iterations for the GAN trained on each dataset (a), (b), (c) and (d) respectively is 9.53%, 1.19%, 5.24% and 2.25%. The predicted v_{bl} errors using this method are small, however, on inspection the predictions in Figure 6.3 fail to produce samples that correlate to a continuous ballistic curve. This can be explained, to an extent, by patterns in the experimental training data. Training set (a) covers the full range of the ballistic curve, however, four points between $400 < v_i < 500$ m/s lie very close together. This disproportionate weighting appears to have affected the GAN's ability to find the trend of the ballistic curve and as such no samples are generated at the ballistic limit. Conversely, half the training data occur at $v_r = 0$ for (c) and predictions get 'stuck' to the x axis; Figure 6.3 shows points with $v_r = 0$ for v_i up to 900 m/s which is not representative of an actual ballistic response. GAN predictions for dataset (b) in Figure 6.3 are the most promising of the test cases with the points generated by the model trained for $1,000,000$ iterations following the ballistic curve more consistently than the others. This outcome is not surprising, given that experimental test data for (b) is well distributed across the ballistic curve in comparison to (a), (c) and (d). The GAN arguably delivers the second best representation of the ballistic curve for dataset (d) in Figure 6.3, however the samples generated by the GAN are again distributed closely to the data in the training set - rather than generalising the ballistic curve itself.

6.2.1 Training evaluation and discussion

The optimum outcome of the GAN's 'zero-sum game' is a consistent prediction by D for both outputs; i.e. the homogenisation of $D(\mathbf{X})$ and $D(G(\mathbf{Z}))$. This is because the GAN's objective is to find an equilibrium between two competing neural networks; G and D. G aims to produce fake data that is similar enough to the real data to fool D, while D aims to accurately distinguish between real and fake data. This competition is what establishes the zero-sum game, meaning that any gain made by one network must be balanced by a loss from the other network. In other words, the gains and losses of G and D cancel each other out and result in the zero-sum game. The objective of the GAN training process is therefore to reach a Nash Equilibrium, a state where both G and D are performing optimally and neither can improve their performance without hurting the other. At this point, G is able to produce realistic synthetic data that is indistinguishable from real data, and D is able to accurately distinguish between real and fake data.

Figure 6.4 presents data collected on performance for all four GANs over training iterations.

Plots on the left hand side show the percentage error of the v_{bl} throughout the training process. The plots on the right hand side present the predictions of the D network: $D(\mathbf{X})$ and $D(G(\mathbf{Z}))$ during training. 10 point moving averages are plotted on both graphs to help interpret the trends of the D prediction. Convergence at an optimum solution is observed for the GAN trained on dataset (b), which shows the most accurate predictions in Figure 6.3b. This is supported in Figure 6.4d where $D(\mathbf{X})$ and $D(G(\mathbf{Z}))$ appear to homogenise after approximately $200,000$ iterations. This trend remains consistent throughout the rest of the training, with $D(\mathbf{X})$ and $D(G(\mathbf{Z}))$ showing very similar trends at $900,000$ iterations. Accurate convergence is corroborated by Figure 6.4c, which shows a decrease in error at around $200,000$ iterations that is maintained for the remainder of training. Homogenisation of $D(\mathbf{X})$ and $D(G(\mathbf{Z}))$ is also observed in Figure 6.4f at approximately $600,000$ iterations. It is at this point that G begins to produce samples at $v_r = 0$, where four out of the seven training instances occur, as shown in Figure 6.3c. This suggests that the GAN begins converging to an optimal solution, however, $D(\mathbf{X})$ and $D(G(\mathbf{Z}))$ lose stability after $800,000$ iterations. This outcome is not surprising, as experimental dataset (c) has arguably the least reliable ballistic curve trend of the four variations. Despite having a consistently low prediction error through training, plots of $D(\mathbf{X})$ and $D(G(\mathbf{Z}))$ in Figure 6.4h appear to never homogenise. The case is similar for Figure 6.4b but with a larger prediction error. These inconsistencies can largely be attributed to clusters in the training data, which make it difficult for the GAN to converge on a trend, despite generating samples that lie close to the ballistic curve.

6.3 Conclusion

This section applied the GAN model from Chapter 4 to experimental ballistic datasets. A total of 4 GAN models were trained on 4 experimental training sets carried out by Costas et al. [Costas et al., 2021] which correspond to a steel bullet core and AP bullet impacting both an as-printed plate and a heat-treated plate. In each case, the GAN prediction error of v_{bl} was less than 10%. Some of the limitations of GAN models were demonstrated through this study. It was noted that the output from the GAN is heavily dependent by the quality of the dataset, synthetic samples generated by the GAN models tended to cluster around the examples from the training set. Whilst this is expected, the GAN model was unable to generalise the ballistic curve. Training GAN networks is also a challenging task, as demonstrated by the plots in Figure 6.4. This study therefore highlights three key weaknesses when applying the standard GAN model on experimental data:

1. Difficulties generalising synthetic ballistic samples across a wide range of impact velocities
2. Generated samples demonstrate a lack of understanding of the physical mechanisms that govern ballistic data

3. The adversarial nature of GAN models can often make training unstable.

The final chapter presents a methodology to address these issues by implementing what is known as a VAE-GAN. The VAE-GAN's have been used in the literature to stabilise the training process of GANs and have demonstrated the ability to produce synthetic data of higher quality and greater accuracy [Arjovsky and Bottou, 2017]. Specifically, the methodology proposes a way to address the GANs lack of understanding of the physical mechanisms that govern ballistic data through the use of Physics-Informed Machine Learning Models (PIML) that incorporate physical principles and constraints into machine learning models such that the model can generalise better when trained on new data and produce more accurate and physically meaningful predictions.

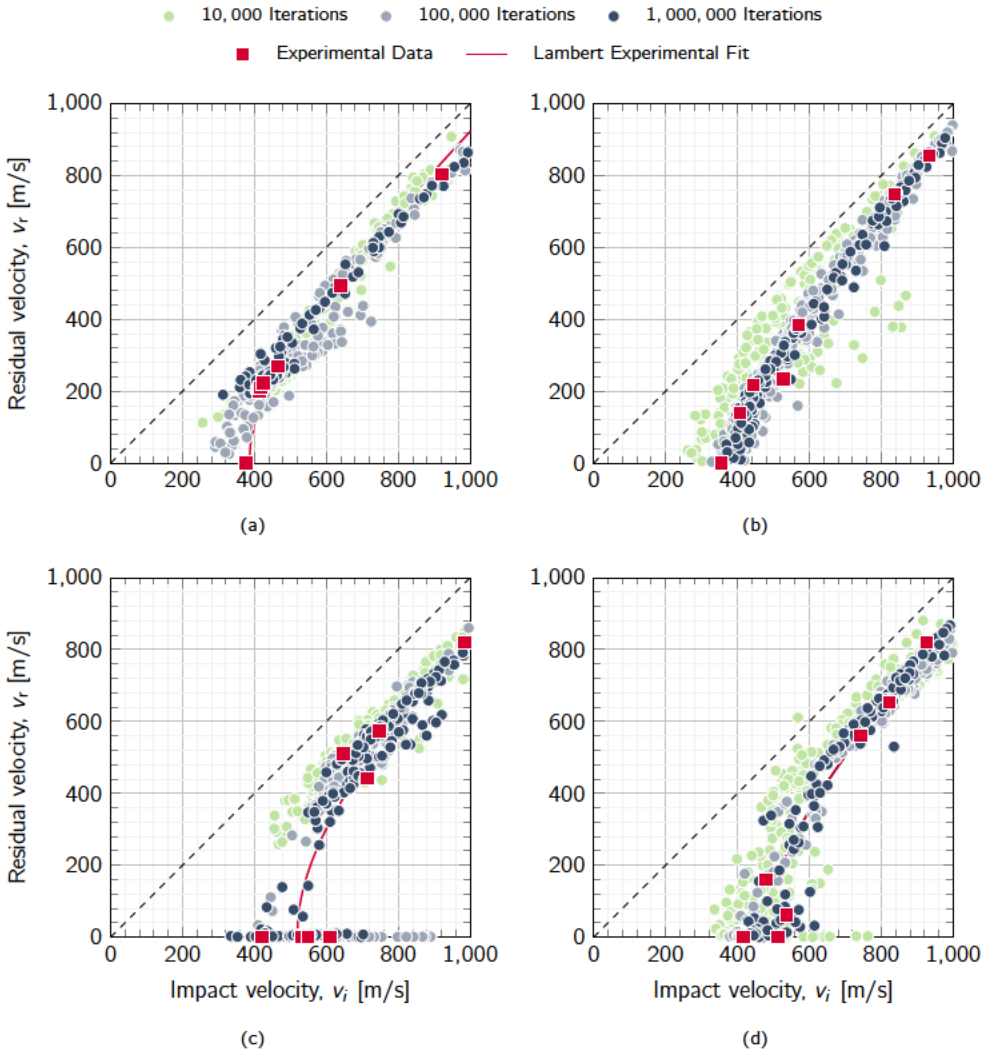


Figure 6.3: Iterative GAN predictions of ballistic curves for experimental tests on maraging steel plates [Costas et al., 2021]: (a) as-printed; core only, (b) as-printed; full bullet, (c) heat treated; core only, and (d) heat treated; full bullet.

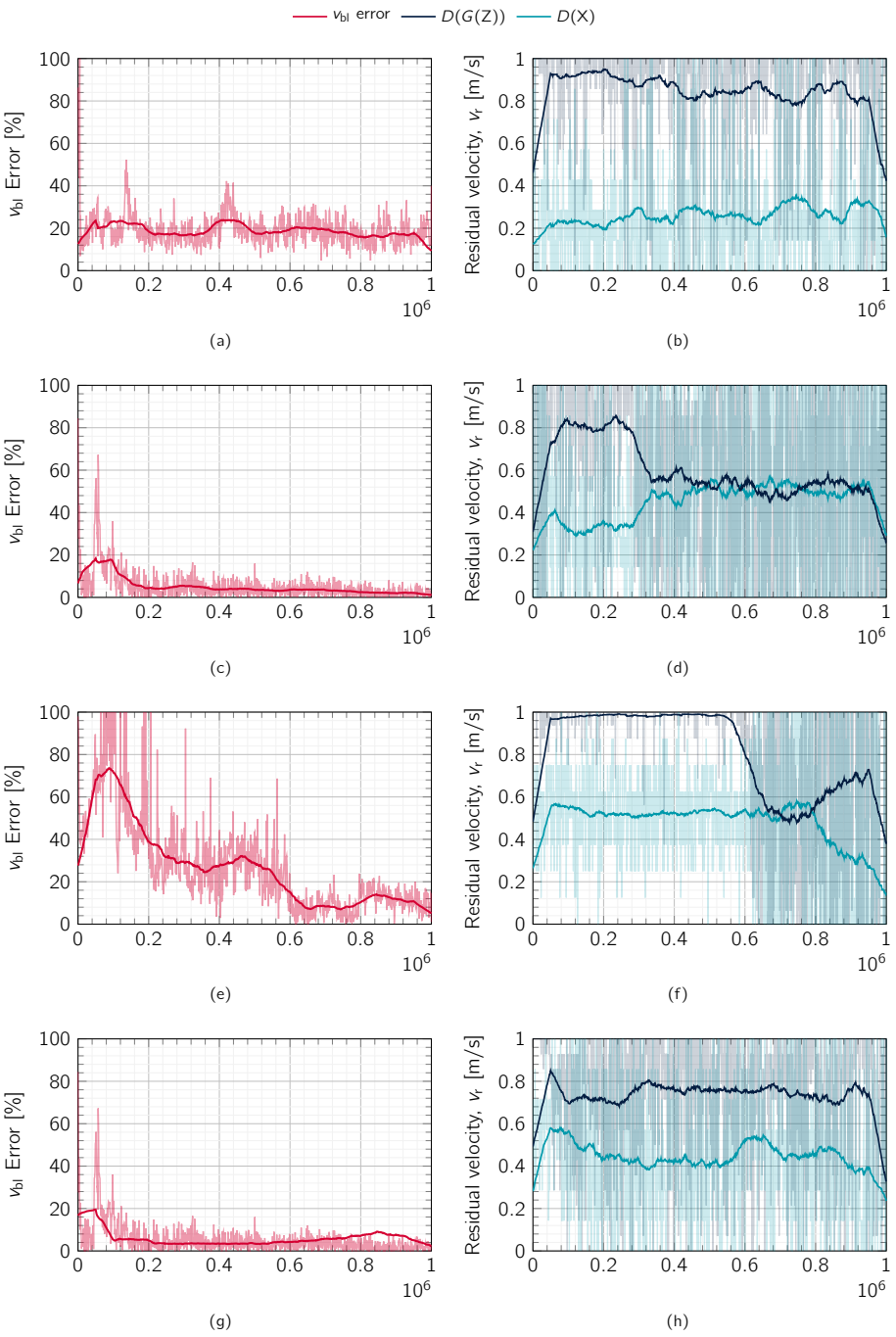


Figure 6.4: GAN output over training iterations with 100 point moving averages for datasets (a), (b), (c), and (d) referred to in Table 6.1 and Figure 6.3: (i) percentage error of ballistic limit velocity estimate, (ii), discriminator prediction of 'real', $D(X)$, and 'fake', $D(G(Z))$, data points.

Chapter 7

Future work and physics informed machine learning

PIML is an emerging field that combines physics-based modeling with data-driven machine learning techniques. PIML models are designed to learn from available data, while still respecting known physical laws and constraints. These models have the potential to improve the accuracy, efficiency, and interpretability of predictions in a variety of scientific and engineering applications. Traditional machine learning models are often limited by the quality and quantity of available training data, and may not be able to capture the underlying physical principles governing a system. On the other hand, physics-based models rely on assumptions and simplifications that may not fully capture the complexity of real-world systems. PIML models aim to overcome these limitations by leveraging both data-driven and physics-based approaches. By incorporating known physical laws and constraints into machine learning models, PIML can improve the accuracy and generalisability of predictions, even when limited data is available. This can lead to faster and more reliable predictions, as well as better insights into the underlying physical processes. PIML models have already shown promising results in a range of applications, including fluid dynamics, materials science, and climate modeling [Karniadakis et al., 2021]. They have the potential to transform many areas of scientific research and engineering, by enabling more accurate and efficient simulations and predictions of complex systems.

Key issues around GANs from the previous chapters include, difficulties to train, dependency on training data etc. This chapter considers how PIML can be used to support the use of GANs to generate better predictions. The ballistic response of any material is governed by certain truths; for example, we know that Vr cannot exceed vi, we know that for velocities less than blv, the residual velocity should be zero. We know that for higher impact velocities, the vr will tend towards vi in an asymptotic fashion. There are analytical models, such as the Lambert

model, that accurately describe this phenomenon and these characteristics. If there is a way that we could embed such Physics into our model, and *inform* it provide additional constraints as to how it should behave - we might be able to achieve better results that are both more accurate and reliable.

This chapter first introduces another type of generative network known as a Variational Autoencoder (VAE) and demonstrates how they can learn from physics-based models such as the Lambert model from a simple test case. A methodology is then proposed that suggests how a VAE can be integrated with the GAN model in a way that has been shown to improve the stability and quality of generated samples [Arjovsky and Bottou, 2017].

7.1 Variational Autoencoders

In recent years, there has been a significant interest in developing efficient techniques for unsupervised learning of high-dimensional data. One popular approach is the use of Autoencoders (AEs), which learn a compressed representation of the latent space, by encoding it into a lower-dimensional space and subsequently decoding it back into the original space. Autoencoders have shown remarkable success in various applications, such as image and speech recognition [Hinton and Salakhutdinov, 2006, Masci et al., 2011], natural language processing [Shixin et al., 2022] and anomaly detection [Saeedi and Giusti, 2023]. However, traditional autoencoders suffer from a major drawback: they do not impose any constraint on the latent space, which can lead to overfitting and poor generalization. To address this issue, Variational Autoencoders (VAEs) were introduced as a probabilistic extension of AEs, which impose a prior distribution on the latent space and learn to encode the input data into a distribution, rather than a fixed point [Kingma and Welling, 2013]. VAEs have shown superior performance in generating high-quality images and have been used in many applications such as image and video generation [Larsen et al., 2015], data augmentation [Xie et al., 2020], and anomaly detection [Schlegl et al., 2017]. They have also been used in combination with other deep learning techniques such as generative adversarial networks (GANs) to improve their stability and quality of the generated samples.

7.1.1 VAE Architecture

In this short study, a VAE is used to generate ballistic data representative of that of a training set from the Lambert analytical model. The VAE can be interpreted as an encoder and a decoder model that are joined together at a bottleneck layer and trained as a single entity. The bottleneck layer is analogous to the latent input to the Generator in the previous GAN examples. Both the encoder and decoder consists of a single fully-connected hidden layer that consists of

1,000 nodes. The output from the encoder hidden layer enters the bottleneck layer. In this example, the bottleneck layer consists of 10 nodes and the output from the bottleneck then passes straight into the decoder layer. The encoder receives training data $\mathbf{X}$ and the decoder outputs a reconstruction of that data $\tilde{\mathbf{X}}$. Once trained, the encoder model can be used in isolation to generate new samples. This is done by passing a latent input $\mathbf{Z}$ into the bottleneck layer (input) and yielding the samples generated from the decoder model in the output.

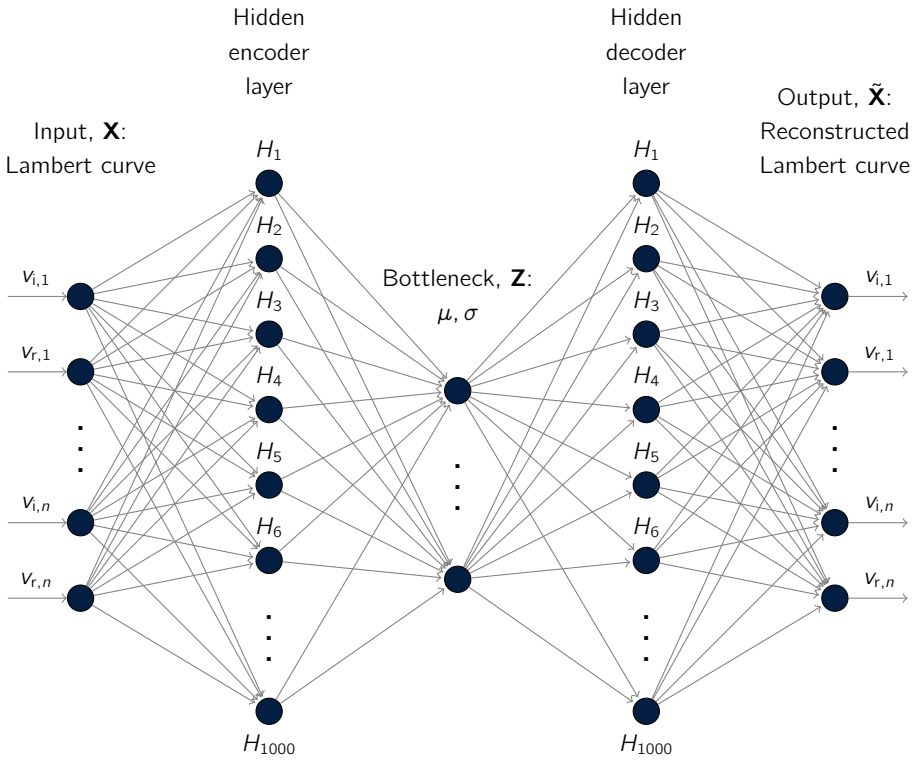


Figure 7.1: Architecture of the VAE used to generate ballistic samples after training on data from the Lambert model.

The mathematical operations of a VAE can be described by a probabilistic framework [Kingma and Welling, 2014, Doersch, 2016]. The bottleneck is a vector of latent variables, $\mathbf{Z}$, which comprise means,

The VAE was trained on 100 Lambert curves, shown in Figure 7.2a, with Lambert parameters a and p set equal to 1 and 2 respectively. Each curve was made up of 100 (v_i, v_r) ballistic data points, resulting in a input vector $\mathbf{x}$ of dimensionality 200. Before input to the VAE, training data was first scaled between $[0, 1]$ in order to comply with computations inside the VAE. To regularise the training of the hidden layers a dropout rate of 0.1 was specified for

both the encoder and decoder. The activation functions for all hidden layers were LReLU functions with $\alpha = 0.2$. The reconstruction loss was calculated using the same function as the GAN loss, binary cross entropy loss, while total VAE loss is the sum of reconstruction loss and Kullback-Leibler divergence. The VAE was trained for 100 epochs and the total training time was 7 seconds. Once trained, the decoder network was tested individually. Figure 7.2 shows the results of using the decoder model on a latent input $\mathbf{Z}$ and observing the quality of the learned distribution $P(\mathbf{X}|\mathbf{Z})$ for the observed training data.

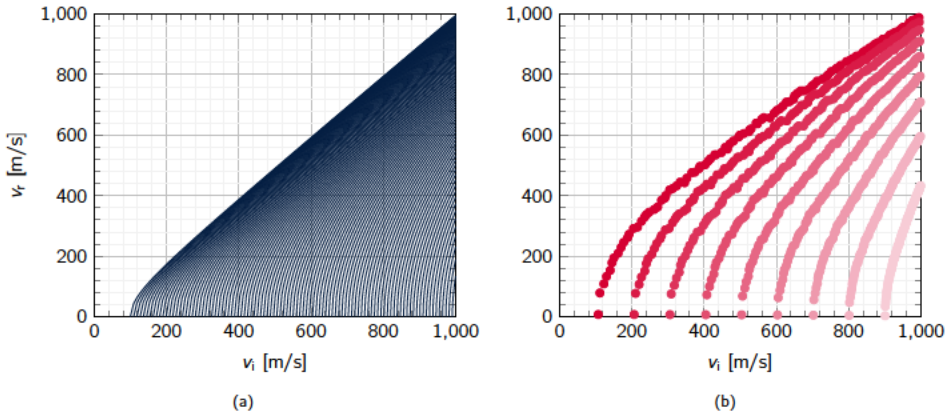


Figure 7.2: VAE training data and results: (a) 100 Lambert training curves and (b) 100-point reconstructions of seven Lambert curves from latent decoder input.

For a range of Lambert models from 300 to 900 m/s, the decoder is able to successfully reconstruct the ballistic curve with very minor scatter in the results. The v_{bl} and corresponding errors were obtained by fitting a Lambert curve to the reconstructed models and are listed in Table 7.1. The samples generated by the decoder model show very good agreement with the Lambert model with all errors below 2%.

Table 7.1: Functions and parameters used in the VAE to pass information through the network during training.

Lambert v_{bl} [m/s]	300	400	500	600	700	800	900
VAE v_{bl} [m/s]	305.51	404.72	503.98	602.90	701.71	800.73	900.15
Error [%]	1.84	1.18	0.78	0.48	0.24	0.09	0.02

7.1.2 Discussion

The benefit of this approach is that now we have a methodology to produce a trained decoder model that can be used to generate ballistic samples from the probability distribution that it has

learned from the Lambert training data. Now when a latent input $\mathbf{Z}$ is passed into the decoder model, samples are generated that are representative of what *real* ballistic data is expected to look like. Since the Lambert analytical model is a generalisation of the Recht-Ipson model that describes v_r after complete perforation in relation to v_i and v_{bl} based on conservation laws of energy and momentum, the encoder model has therefore been informed by a physical model that is used regularly within the study of ballistic impact [Børvik et al., 1999a, Costas et al., 2021]. There is therefore an opportunity to leverage the pre-trained encoder model in a typical GAN network in a fashion that utilises the benefits of transfer learning.

Transfer learning is a powerful technique in machine learning where a model trained on one dataset can be used to solve a different but related task on a different dataset. This technique has many benefits, particularly in cases where data is limited, as in the case of ballistic testing where it is expensive to establish a large training dataset due to the destruction of samples across a wide range of impact velocities. Transfer learning allows a model to leverage the knowledge learned from one dataset to learn features that are relevant to another dataset, such as $v_r = 0$ for $v_i < v_{bl}$ and v_r cannot exceed v_i. Another benefit of transfer learning is improved generalisation, where a model can use knowledge learned from a large dataset to improve its performance on a smaller dataset. For example, the Lambert model can be used to generate a large dataset to pre-train the decoder model, which can then be retrained on the smaller, but newer, experimental dataset. Additionally, transfer learning can speed up the training process by initialising the trainable parameters, *weights*, of the decoder model with pre-trained weights, which can reduce training time and improve the convergence of the model [Liu and Shi, 2020]. A schematic of this configuration is presented in Figure 7.3. Note that the decoder presented in Section 7.1.1 outputs 50 samples for a given input $\mathbf{Z}$, it would then be possible to isolate one random generated sample to pass into the discriminator network or restrict further updates to the weights of the encoder model until the discriminator has evaluated each of the 50 samples.

GANs are powerful generative models that can learn to generate fake samples that are indistinguishable from real data by a discriminator model. However, they can be challenging to train. In Chapter 6, the GAN model from Chapter 4 was tested on real experimental data. Whilst the results were okay, the models were heavily dependent on the quality of the experimental data provided and struggled to generate samples across a spectrum of impact velocities - often the generated data would cluster around the samples present in the training data. This instability is caused by the non-convex nature of the optimisation problem that GANs attempt to solve, as well as the delicate balance between the performance of the generator and discriminator networks [Sixt et al., 2018]. To address the instability of the GAN, VAEs are used to regularise the latent space. Specifically, this is done by the replacement of the GAN generator a VAE. The GAN's discriminator is then used to evaluate the generated samples by comparing them to real data in the same way described in Chapter 4. By using a VAE to regularise the generators latent space, it is the intention that the VAE-GAN is able to learn a more structured and meaningful representation of the data which results in more stable and

higher-quality generated samples. This type of architecture has been successfully applied already to various domains within the literature. For example, in image synthesis, VAE-GANs have been used to generate high-resolution images with sharper and more detailed features compared to traditional GANs [Larsen et al., 2015].

This methodology could also be extended in a similar fashion to that described in Chapter 5. Whilst cGANs provided an opportunity to condition traditional GAN networks on additional information, such as class labels, a conditional Variational Autoencoder Generative Adversarial Network (cVAE-GAN) is a model that allows the same adaptation of a traditional VAE-GANs. In a cVAE-GAN, the encoder and decoder networks of the VAE are conditioned on additional information, such as class labels, in addition to the input data. The discriminator network of the GAN is also conditioned on the same information. This conditioning allows the network to generate samples that are conditioned on specific attributes or classes. There is therefore an opportunity for additional research to consider a physics-informed cVAE-GAN model to generate additional samples for classes that do not exist in the training set. Similar to the methodology proposed in Chapter 5, where a class might correspond to the thickness of the target sample, predictions could be made for classes within the domain of the training set and beyond.

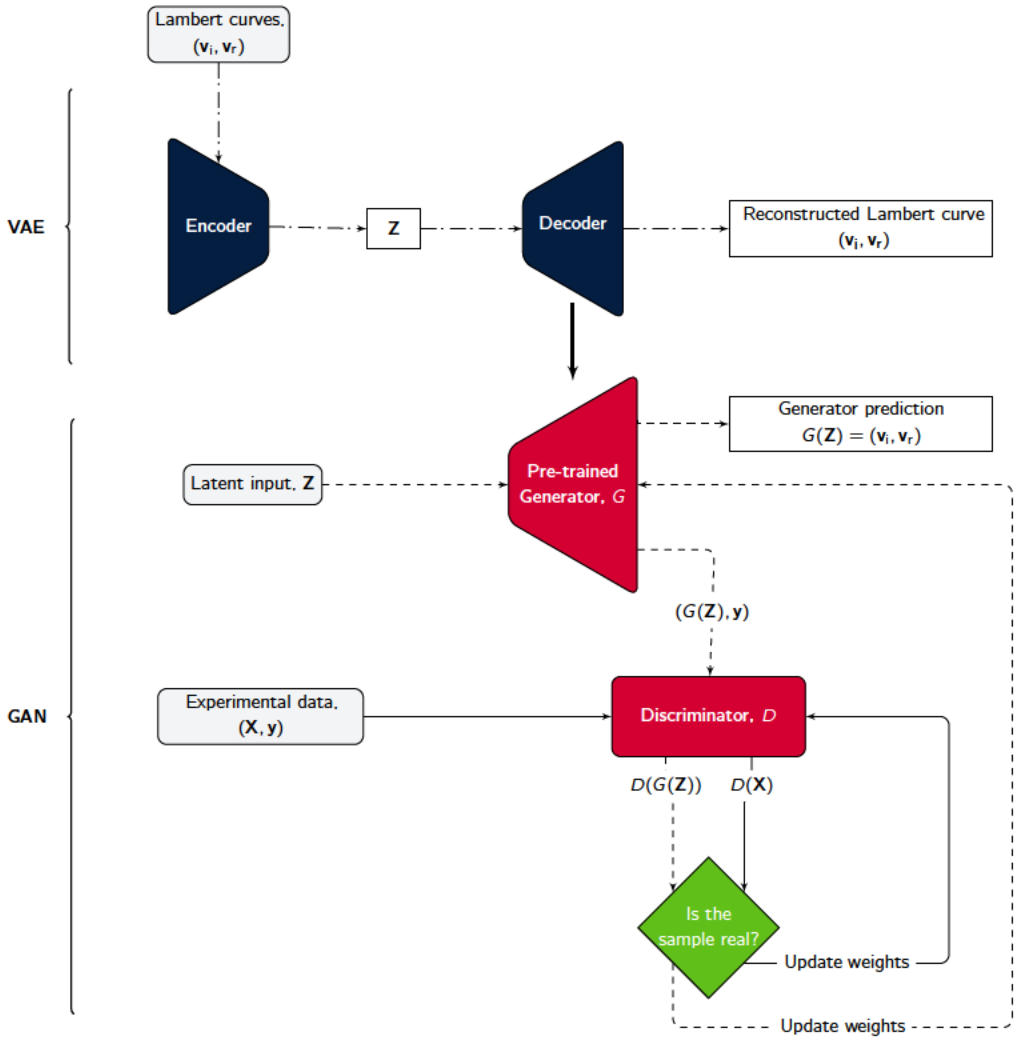


Figure 7.3: Flowchart of processes for a fine tuned GAN, with the generator, G, replaced by a pre-trained VAE decoder network. The solid line represents discriminator training, the dashed line represents generator training, and the dot-dashed line represents VAE training.

Chapter 8

Thesis Concluding Remarks

Having discussed the possibilities of machine learning within the ballistic domain, specifically that of generative networks, some concluding remarks are given. Firstly, the significance and implications of this work are discussed. Lastly, the limitations of the proposed methods are given along with recommendations for future work.

8.1 Significance and Implications

The aim of this research was to investigate whether ML methods can be applied to predict the response of materials and structures to ballistic impact and facilitate the design of armour systems. Conventionally, approaches to understand the response to ballistic impact typically revolve around experimental tests, whereby the material or structure of interest is subjected to impact by a projectile in a controlled environment across a range of impact velocities. However, the need for specialist equipment such as high-speed cameras combined with the destructive nature of experimental testing incurs large costs for each of the experimental processes involved. This cost is magnified when evaluating the response of complex materials such as composites that are also expensive to fabricate or source. High strain rate material characterisation campaigns often accompany ballistic testing in order to develop and calibrate material models for numerical simulations of the penetration and perforation processes. Numerical tools, such as the FE method, play an important role by filling the gaps left sparse by experimental results and the parameters of the numerical models are adjusted such that an agreement with the experimental data is met. This process is time consuming and adds additional cost to the campaign. There is therefore an opportunity to explore whether ML models can be implemented to learn directly from ballistic data and make subsequent predictions regarding the ballistic

performance of a subject.

This thesis has shown that trained ML models can be used to supplement ballistic datasets. Chapter 4 proposed a GAN architecture that was trained on 3 ballistic datasets of degrading quality and the trained models were able to generate additional samples representative of the training set. More importantly, the generated samples were used to predict the v_{bl} with an error of less than 5% in all cases. The proposed GAN architecture was able to learn directly from the ballistic training set and, unlike numerical methods, there is no requirement for additional tuning in order to improve the agreement between the generated samples and those which are expected. Instead, the fine tuning was done during the training process and is governed by the elected optimiser and loss function that optimised the trainable parameters of the network. An important limitation with the traditional GAN network, is that its output is limited to samples from the same distribution that it was trained on as there is no local control over the output. As such, when the trained GAN is used to generate a new sample or make a new prediction, that new prediction is selected from the distribution that it has learned and governed by the latent input. The trained GAN can be used to make many predictions in order to assess the scope of the learned distribution. In the case of the GAN trained in Chapter 4, this entire distribution corresponds to a complete ballistic curve. What the GAN cannot do, however, is make predictions outside of the domain that it was trained on. For instance, if the trained GAN was trained on a ballistic dataset that corresponds to the perforation of a 10 mm thick aluminium plate by a blunt, cylindrical projectile, it would not be possible to use the model to predict the ballistic response of a thicker plate to the same loading scenario. The GAN is therefore limited to generating new samples that are representative of the data that it was trained on.

Chapter 5 however, implemented a variant of the GAN known as the cGAN. The cGAN addresses a key limitation of conventional GANs where there is limited governance over the output. cGANs can be conditioned by extra information during training such as class labels or data from other modalities to grant additional control over the output. In this study, a proposed cGAN architecture was trained on a multi-class ballistic dataset containing 10 classes where each class corresponded to a ballistic curve with a different v_{bl}. A total of 5 models were trained on training sets consisting of 5, 10, 15, 20 and 25 ballistic samples within each class. The cGAN model requires a class label input and the output is based on what the model has learned to associate with that particular label during training. To that end, for integer class labels $0-9$, all cGAN models successfully predicted the v_{bl} with an error of less than 4.12%. More interestingly, it was also found that for non-integer class labels between $0-9$, the v_{bl} predictions were similar despite not explicitly appearing in the training set. Moreover, each of the cGAN models were challenged to generate new ballistic samples for class labels that exist beyond the scope of the training set for class labels between $9-20$. It was found that four of the models were able to predict the v_{bl} with an error of less than 1.5%. This has a direct application for use within material characterisation campaigns as the cGAN can be

used to generate samples for classes that the model was not explicitly trained on. Unlike the conventional GAN model, the cGAN could be trained on a multi-class ballistic data set where each class refers to the ballistic response of a material at a different thickness and intermediate class labels could be passed into the network to make predictions for intermediate thicknesses.

It should be noted however, that each prediction from the generative models in this thesis returns a single ballistic sample that contains the v_i and the corresponding v_r. Therefore the output from the GAN offers ballistic samples only, and no additional information is attained regarding the method of perforation, plugging or any auxiliary information regarding the failure mechanics of the target plate. This is an important distinction between the proposed generative methodology and the numerical methods that are currently being used. Numerical methods are still useful as they yield much more information regards to predicting the actual fracture mechanics and failure modes of ballistic impact.

Chapter 6 applied the GAN model presented in Chapter 4 to an experimental dataset conducted by Costas et al. [Costas et al., 2021]. This study considered 4 test cases on AM maraging steel both as-printed and heat treated subject to impact from both a full AP bullet and just the core. In each case, the prediction error of the v_{bl} was less than 10%. However, it was found that the GAN struggled to generalise synthetic ballistic samples across a wide range of impact velocities as the generated samples clustered close to the data present in the training set. Whilst this is to be expected, it reduces the scope of application for this method. The study also highlights the difficulties associated with training GAN networks in practice.

8.2 Contribution to knowledge

Overall, this thesis presents a novel approach to train cGAN networks on incomplete material characterisation campaigns that has not been previously considered in the literature. A trained cGAN model can be used to (1) generate additional samples that belong to a desired class (or plate thickness) and (2) subsequently make v_{bl} predictions for the desired classes. The cGAN network would simply require a training set made from the available experimental ballistic data and, by using the methodology presented in Chapter 5, the trained model can be used to generate new ballistic samples immediately. This offers a valuable benefit over numerical methods as they can generate additional samples and make new predictions quickly irrespective of the type of material used or the particular impact conditions. Depending on the complexity of the experiment, FE models can take substantial time to set up and validate the model such that its output aligns with that of the experiments. In addition, for each new ballistic sample predicted via the numerical model, an entire simulation must be completed which adds additional time and cost. This approach however, allows for quick and accurate predictions to be made regarding the ballistic response of a material. The predictions of which could be used

to influence the design process such that loading configurations with undesirable qualities can be identified and eliminated sooner and conversely favourable configurations can be identified and used to speed up the development of prototype armour systems.

8.3 Future Work

This thesis utilised ballistic datasets that were curated via analytical models to demonstrate the potential of the methodology. Whilst the performance of the GAN was considered on experimental data, it would be worthwhile to also test the cGAN architectures directly on experimental datasets. The extensive ballistic research performed by Børvik et al. [Børvik et al., 1999a, Børvik et al., 2001b, Børvik et al., 2003, Børvik et al., 2005] includes the results from material characterisation campaigns at different plate thicknesses. The data could be structured into a multi-class ballistic training set, as shown in Chapter 5, where each thickness corresponds to a different class of ballistic data. Ballistic data that corresponds to the results from one of the material thickness campaigns could be omitted from the training set to form test cases for the generated samples from the trained cGAN model to be compared against.

In addition, it would be beneficial to stress-test the performance of the cGAN network and consider its robustness against training sets of varying quality. The study presented in Chapter 5 considers a multi-class training set that consists of 10 classes and a comparison is made between models based on the number of samples that are present within each class. In practice, it cannot be guaranteed that 10 classes of ballistic data are available to form an equivalent training set. Instead, additional research should be conducted to determine the limitations of the methodology and find the minimum number of classes required to achieve acceptable results. To that end, multiple cGAN networks could then be trained on separate ballistic datasets that contain a different number of classes and a varying number of samples within each class.

Whilst the application of the cGAN was the biggest achievement of this thesis, there is an opportunity to explore the implementation of transfer learning in this domain on the standard GAN specifically. Transfer learning is a machine learning method where a model developed specifically for a task is reused as the starting point for a model on a second task. To that end, a GAN could be trained on an analytical dataset to learn the key features that constitute the ballistic curve. That includes: (1) a region below the v_{bl} where all impact velocities return a residual velocity of 0, (2) a distinct v_{bl} location, where impact velocities greater than v_{bl} return a residual velocity that is greater than 0 and (3) the residual velocity never exceeds the impact velocity. Once the model has learned the key features of a ballistic curve, it could then be re-trained and conditioned directly on ballistic samples produced by experiments. The intention would be to further reduce the number of samples required to train the model, given that the pre-trained model already has an understanding of the key features that constitute

the ballistic curve. This could allow for the GAN model to be more flexible when training on ballistic datasets and remove the requirement for the training set to contain samples that exist near the v_{bl}. The preferred implementation of this would implement physics-informed machine learning models as described in Chapter 7. This approach would consider a VAE-GAN where the G model of the GAN is replaced by a decoder model that was pre-trained on a ballistic training set such as the Lambert analytical model. The Lambert model is a generalisation of the Recht-Ipson model that describes v_r after complete perforation in relation to v_i and v_{bl} based on conservation laws of energy and momentum. The intention is that pre-training the VAE-GAN model on this data could help to both generalise the synthetic data produced from the model whilst also improving its quality. Similar to the cGAN model presented in Chapter 5, there is also an opportunity to consider a cVAE-GAN model to make predictions for classes that do not exist in the training set and can therefore make a true contribution to material characterisation campaigns.

Bibliography

[Abladey and Braimah, 2014] Abladey, L. and Braimah, A. (2014). Near-field explosion effects on the behaviour of reinforced concrete columns: a numerical investigation. *International Journal of Protective Structures*, 5(4):475–499.

[Aggarwal et al., 2021] Aggarwal, A., Mittal, M., and Battineni, G. (2021). Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, 1(1):100004.

[Ahmadian, 2016] Ahmadian, A. S. (2016). *Numerical Models for Submerged Breakwaters*. Elsevier.

[Al-Bayti, 2017] Al-Bayti, A. (2017). *Vulnerability of Reinforced Concrete Columns to External Blast Loading*. PhD thesis, Université d'Ottawa/University of Ottawa.

[Albarqouni et al., 2016] Albarqouni, S., Baur, C., Achilles, F., Belagiannis, V., Demirci, S., and Navab, N. (2016). Aggnet: Deep learning from crowds for mitosis detection in breast cancer histology images. *IEEE Transactions on Medical Imaging*, 35(5):1313–1321.

[Alber et al., 2019] Alber, M., Buganza Tepole, A., Cannon, W., De, S., Dura-Bernal, S., Garikipati, K., Karniadakis, G., Lytton, W., Perdikaris, P., Petzold, L., and Kuhl, E. (2019). Integrating machine learning and multiscale modeling—perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences. *npj Digital Medicine*, 2.

[Ali et al., 2017] Ali, M. W., Mubashar, A., Uddin, E., Haq, S. W. U., and Khan, M. (2017). An experimental and numerical investigation of the ballistic response of multi-level armour against armour piercing projectiles. *International Journal of Impact Engineering*, 110:47–56. Special Issue in honor of Seventy Fifth Birthday of Professor N. K. Gupta.

[Almustafa and Nehdi, 2022] Almustafa, M. K. and Nehdi, M. L. (2022). Machine learning model for predicting structural response of rc columns subjected to blast loading. *International Journal of Impact Engineering*, 162:104145.

[Arjovsky and Bottou, 2017] Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks.

[Ashby and Jones, 2019] Ashby, M. F. and Jones, D. R. H. (2019). *Engineering Materials 2: An Introduction to Microstructures, Processing and Design*. Butterworth-Heinemann, 4th edition.

[Atkins et al., 1998] Atkins, A., Afzal khan, M., and Liu, J. (1998). Necking and radial cracking around perforations in thin sheets at normal incidence. *International Journal of Impact Engineering*, 21(7):521–539.

[Awerbuch and Bodner, 1974] Awerbuch, J. and Bodner, S. (1974). Experimental investigation of normal perforation of projectiles in metallic plates. *International Journal of Solids and Structures*, 10(6):685–699.

[Backman and Goldsmith, 1978] Backman, M. E. and Goldsmith, W. (1978). The mechanics of penetration of projectiles into targets. *International Journal of Engineering Science*, 16(1):1–99.

[Baek et al., 2020] Baek, S., Kim, K. I., and Kim, T.-K. (2020). Weakly-supervised domain adaptation via gan and mesh model for estimating 3d hand poses interacting objects. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6120–6130.

[Baker et al., 2019] Baker, N., Alexander, F., Bremer, T., Hagberg, A., Kevrekidis, Y., Najm, H., Parashar, M., Patra, A., Sethian, J., Wild, S., Willcox, K., and Lee, S. (2019). Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence.

[Basaran, 2017] Basaran, G. (2017). Numerical study of high velocity impact response of vehicle armor combination using ls dyna.

[Battineni et al., 2020] Battineni, G., Sagaro, G. G., Chinatalapudi, N., and Amenta, F. (2020). Applications of machine learning predictive models in the chronic disease diagnosis. *Journal of Personalized Medicine*, 10(2):21.

[Ben-Dor et al., 2002] Ben-Dor, G., Dubinsky, A., and Elperin, T. (2002). On the lambert–jonas approximation for ballistic impact. *Mechanics Research Communications*, 29(2):137 – 139.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Bobbili et al., 2020] Bobbili, R., Ramakrishna, B., and Madhu, V. (2020). An artificial intelligence model for ballistic performance of thin plates. *Mechanics Based Design of Structures and Machines*, 0(0):1–12.

[Bode et al., 2019] Bode, M., Gauding, M., Lian, Z., Denker, D., Davidovic, M., Kleinheinz, K., Jitsev, J., and Pitsch, H. (2019). Using physics-informed super-resolution generative adversarial networks for subgrid modeling in turbulent reactive flows.

[Boden, 2014] Boden, M. A. (2014). *GOFAI*, page 89–107. Cambridge University Press.

[Bresciani et al., 2015] Bresciani, L., Manes, A., and Giglio, M. (2015). An analytical model for ballistic impacts against plain-woven fabrics with a polymeric matrix. *International Journal of Impact Engineering*, 78:138–149.

[Bresciani et al., 2016] Bresciani, L., Manes, A., Romano, T., Iavarone, P., and Giglio, M. (2016). Numerical modelling to reproduce fragmentation of a tungsten heavy alloy projectile impacting a ceramic tile: Adaptive solid mesh to the sph technique and the cohesive law. *International Journal of Impact Engineering*, 87:3–13. SI: Experimental Testing and Computational Modeling of Dynamic Fracture.

[Brock et al., 2019] Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale gan training for high fidelity natural image synthesis.

[Brown, 1986] Brown, S. J. (1986). Energy Release Protection for Pressurized Systems. Part II: Review of Studies Into Impact/Terminal Ballistics. *Applied Mechanics Reviews*, 39(2):177–201.

[Bush, 2019] Bush, V. (2019). As we may think.

[Børvik et al., 2005] Børvik, T., Clausen, A. H., Eriksson, M., Berstad, T., Sture Hopperstad, O., and Langseth, M. (2005). Experimental and numerical study on the perforation of aa6005-t6 panels. *International Journal of Impact Engineering*, 32(1):35–64. Fifth International Symposium on Impact Engineering.

[Børvik et al., 2009] Børvik, T., Dey, S., and Clausen, A. (2009). Perforation resistance of five different high-strength steel plates subjected to small-arms projectiles. *International Journal of Impact Engineering*, 36(7):948–964.

[Børvik et al., 2001a] Børvik, T., Hopperstad, O., Berstad, T., and Langseth, M. (2001a). A computational model of viscoplasticity and ductile damage for impact and penetration. *European Journal of Mechanics - A/Solids*, 20(5):685–712.

[Børvik et al., 2003] Børvik, T., Hopperstad, O. S., Langseth, M., and Malo, K. A. (2003). Effect of target thickness in blunt projectile penetration of weldox 460 e steel plates. *International Journal of Impact Engineering*, 28(4):413 – 464.

[Børvik et al., 1999a] Børvik, T., Langseth, M., Hopperstad, O., and Malo, K. (1999a). Ballistic penetration of steel plates. *International Journal of Impact Engineering*, 22(9):855 – 886.

[Børvik et al., 1999b] Børvik, T., Langseth, M., Hopperstad, O., and Malo, K. (1999b). Ballistic penetration of steel plates. *International Journal of Impact Engineering*, 22(9):855–886.

[Børvik et al., 2001b] Børvik, T., Leinum, J. R., Solberg, J. K., Hopperstad, O. S., and Langseth, M. (2001b). Observations on shear plug formation in weldox 460 e steel plates impacted by blunt-nosed projectiles. *International Journal of Impact Engineering*, 25(6):553 – 572.

[Callister Jr. and Rethwisch, 2018] Callister Jr., W. D. and Rethwisch, D. G. (2018). *Materials Science and Engineering: An Introduction*. Wiley, 10th edition.

[Chen et al., 2015] Chen, C., Seff, A., Kornhauser, A., and Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving.

[Chen et al., 2018] Chen, Y., Shi, F., Christodoulou, A. G., Xie, Y., Zhou, Z., and Li, D. (2018). Efficient and accurate mri super-resolution using a generative adversarial network and 3d multi-level densely connected network. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 91–99. Springer.

[Chen et al., 2020] Chen, Z., Wang, C., Wu, H., Shang, K., and Wang, J. (2020). Dmgan: Discriminative metric-based generative adversarial networks. *Knowledge-Based Systems*, 192:105370.

[Chocron Benloulo and Sánchez-Gálvez, 1998] Chocron Benloulo, I. and Sánchez-Gálvez, V. (1998). A new analytical model to simulate impact onto ceramic/composite armors. *International Journal of Impact Engineering*, 21(6):461–471.

[Chu et al., 2020] Chu, Y., Yue, X., Yu, L., Sergei, M., and Wang, Z. (2020). Automatic image captioning based on resnet50 and lstm with soft attention. *Wireless Communications and Mobile Computing*, 2020:1–7.

[Collins et al., 2021] Collins, C., Dennehy, D., Conboy, K., and Mikalef, P. (2021). Artificial intelligence in information systems research: A systematic literature review and research agenda. *International Journal of Information Management*, 60:102383.

[Corbett et al., 1996] Corbett, G., Reid, S., and Johnson, W. (1996). Impact loading of plates and shells by free-flying projectiles: A review. *International Journal of Impact Engineering*, 18(2):141–230.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, 20(3):273–297.

[Costas et al., 2021] Costas, M., Edwards-Mowforth, M., Kristoffersen, M., Teixeira-Dias, F., Brøtan, V., Paulsen, C., and Børvik, T. ((submitted) 2021). Ballistic impact resistance of additive manufactured high-strength maraging steel: an experimental study. *The International Journal of Protective Structures*.

[Courant, 1943] Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49(1):1 – 23.

[Cruz-Roa et al., 2013] Cruz-Roa, A., Arevalo, J., Madabhushi, A., and González, F. (2013). A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection. volume 16, pages 403–10.

[Daugherty, 2018] Daugherty, P. R. (2018). Collaborative intelligence : Humans and ai are joining forces.

[Dey et al., 2006] Dey, S., Børvik, T., Hopperstad, O., and Langseth, M. (2006). On the influence of fracture criterion in projectile impact of steel plates. *Computational Materials Science*, 38(1):176–191.

[Dey et al., 2007] Dey, S., Børvik, T., Teng, X., Wierzbicki, T., and Hopperstad, O. (2007). On the ballistic resistance of double-layered steel plates: An experimental and numerical investigation. *International Journal of Solids and Structures*, 44(20):6701–6723.

[Dieter, 1988] Dieter, G. E. (1988). *Mechanical Metallurgy*. McGraw-Hill, 3rd edition.

[Dimiduk et al., 2018] Dimiduk, D. M., Holm, E. A., and Niezgoda, S. R. (2018). Perspectives on the impact of machine learning, deep learning, and artificial intelligence on materials, processes, and structures engineering. *Integrating Materials and Manufacturing Innovation*, 7(3).

[Doersch, 2016] Doersch, C. (2016). Tutorial on variational autoencoders. Technical report, UC Berkeley. arXiv:1606.05908.

[Duan et al., 2019] Duan, Y., Edwards, J. S., and Dwivedi, Y. K. (2019). Artificial intelligence for decision making in the era of big data – evolution, challenges and research agenda. *International Journal of Information Management*, 48:63–71.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[Dwivedi et al., 2021] Dwivedi, Y. K., Hughes, L., Ismagilova, E., Aarts, G., Coombs, C., Crick, T., Duan, Y., Dwivedi, R., Edwards, J., Eirug, A., Galanos, V., Ilavarasan, P. V., Janssen, M., Jones, P., Kar, A. K., Kizgin, H., Kronemann, B., Lal, B., Lucini, B., Medaglia, R., Le Meunier-FitzHugh, K., Le Meunier-FitzHugh, L. C., Misra, S., Mogaji, E., Sharma, S. K., Singh, J. B., Raghavan, V., Raman, R., Rana, N. P., Samothrakis, S., Spencer, J., Tamilmani, K., Tubadji, A., Walton, P., and Williams, M. D. (2021). Artificial intelligence (ai): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy. *International Journal of Information Management*, 57:101994.

[Elek et al., 2005] Elek, P., Jaramaz, S., and Micković, D. (2005). Modeling of perforation of plates and multi-layered metallic targets. *International Journal of Solids and Structures*, 42(3):1209–1224.

[Elfwing et al., 2017] Elfwing, S., Uchibe, E., and Doya, K. (2017). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning.

[Faghmous and Kumar, 2014] Faghmous, J. and Kumar, V. (2014). A big data guide to understanding climate change: The case for theory-guided data science. *Big data*, 2:155–163.

[Feli et al., 2011] Feli, S., Yas, M., and Asgari, M. (2011). An analytical model for perforation of ceramic/multi-layered planar woven fabric targets by blunt projectiles. *Composite Structures*, 93(2):548–556.

[Feng et al., 2020] Feng, J., Li, W., Ding, C., Gao, D., Shi, Z., and Liang, J. (2020). Numerical and analytical investigations on projectile perforation on steel–concrete–steel sandwich panels. *Results in Engineering*, 8:100164.

[Flores-Johnson et al., 2011] Flores-Johnson, E., Saleh, M., and Edwards, L. (2011). Ballistic performance of multi-layered metallic plates impacted by a 7.62-mm apm2 projectile. *International Journal of Impact Engineering*, 38(12):1022–1032.

[Fras et al., 2019] Fras, T., Colard, L., and Reck, B. (2019). Modeling of ballistic impact of fragment simulating projectiles against aluminum plates.

[Gao et al., 2019] Gao, Y., Li, D., Zhang, W., Guo, Z., Yi, C., and Deng, Y. (2019). Constitutive modelling of the tib2–b4c composite by experiments, simulation and neutral network. *International Journal of Impact Engineering*, 132:103310.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

[Go et al., 2020] Go, T., Lee, S., You, D., and Lee, S. J. (2020). Deep learning-based hologram generation using a white light source. *Scientific reports*, 10(1):1–12.

[Gonzalez-Carrasco et al., 2011] Gonzalez-Carrasco, I., Garcia Crespo, A., Ruíz-Mezcua, B., and Cuadrado, J. (2011). Dealing with limited data in ballistic impact scenarios: An empirical comparison of different neural network approaches. *Appl. Intell.*, 35:89–109.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

[Graves et al., 2013] Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778.

[Gregori et al., 2020] Gregori, D., Scazzosi, R., Nunes, S. G., Amico, S. C., Giglio, M., and Manes, A. (2020). Analytical and numerical modelling of high-velocity impact on multilayer alumina/aramid fiber composite ballistic shields: Improvement in modelling approaches. *Composites Part B: Engineering*, 187:107830.

[Griffin, 1961] Griffin, K. H. (1961). Impact: The theory and physical behaviour of colliding solids.w. goldsmith. arnold, london. 1960. 379 pp. diagrams. 90s. *The Journal of the Royal Aeronautical Society*, 65(606):443–443.

[Grnarova et al., 2018] Grnarova, P., Levy, K. Y., Lucchi, A., Perraudin, N., Goodfellow, I., Hofmann, T., and Krause, A. (2018). A domain agnostic measure for monitoring and evaluating gans.

[Gupta et al., 2021] Gupta, A., Anpalagan, A., Guan, L., and Khwaja, A. S. (2021). Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, 10:100057.

[Gupta et al., 2007] Gupta, N., Iqbal, M., and Sekhon, G. (2007). Effect of projectile nose shape, impact velocity and target thickness on deformation behavior of aluminum plates. *International Journal of Solids and Structures*, 44(10):3411–3439.

[hai Chen et al., 2017] hai Chen, C., Zhu, X., liang Hou, H., bing Tian, X., and le Shen, X. (2017). A new analytical model for the low-velocity perforation of thin steel plates by hemispherical-nosed projectiles. *Defence Technology*, 13(5):327–337.

[Han and Moraga, 1995] Han, J. and Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. In Mira, J. and Sandoval, F., editors, *From Natural to Artificial Neural Computation*, pages 195–201, Berlin, Heidelberg. Springer Berlin Heidelberg.

[He et al., 2015a] He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

[He et al., 2015b] He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

[He et al., 2017] He, Z., Chen, W., Wang, F., and Feng, M. (2017). A kinematic hardening constitutive model for the uniaxial cyclic stress–strain response of magnesium sheet alloys at room temperature. *Materials Research Express*, 4(11):116513.

[Hinton, 2012] Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.

[Hinton and Sejnowski, 1999] Hinton, G. and Sejnowski, T. J. (1999). *Unsupervised Learning: Foundations of Neural Computation*. The MIT Press.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[Hochreiter and Schmidhuber, 1997a] Hochreiter, S. and Schmidhuber, J. (1997a). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Hochreiter and Schmidhuber, 1997b] Hochreiter, S. and Schmidhuber, J. (1997b). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Holmen et al., 2013] Holmen, J., Johnsen, J., Jupp, S., Hopperstad, O., and Børvik, T. (2013). Effects of heat treatment on the ballistic properties of aa6070 aluminium alloy. *International Journal of Impact Engineering*, 57:119–133.

[Holmen et al., 2015] Holmen, J. K., Børvik, T., Myhr, O., Fjær, H., and Hopperstad, O. (2015). Perforation of welded aluminum components: Microstructure-based modeling and experimental validation. *International Journal of Impact Engineering*, 84.

[Holmen et al., 2016] Holmen, J. K., Johnsen, J., Hopperstad, O. S., and Børvik, T. (2016). Influence of fragmentation on the capacity of aluminum alloy plates subjected to ballistic impact. *European Journal of Mechanics - A/Solids*, 55:221–233.

[Horn and Johnson, 1985] Horn, R. A. and Johnson, C. R. (1985). *Matrix Analysis*. Cambridge University Press.

[Horne, 1979] Horne, M. R. (1979). *Plastic Theory of Structures*. Elsevier.

[Huang et al., 2018] Huang, X., Zhang, W., Deng, Y., and Jiang, X. (2018). Experimental investigation on the ballistic resistance of polymer-aluminum laminated plates. *International Journal of Impact Engineering*, 113:212 – 221.

[Isola et al., 2018] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2018). Image-to-image translation with conditional adversarial networks.

[Jaiswal et al., 2020] Jaiswal, D. P., Kumar, S., and Badr, Y. (2020). Towards an artificial intelligence aided design approach: application to anime faces with generative adversarial networks. *Procedia Computer Science*, 168:57–64.

[Jia et al., 2014] Jia, X., xiang Huang, Z., dong Zu, X., hui Gu, X., and qiang Xiao, Q. (2014). Theoretical analysis of the disturbance of shaped charge jet penetrating a woven fabric rubber composite armor. *International Journal of Impact Engineering*, 65:69–78.

[Jin et al., 2017] Jin, Y., Zhang, J., Li, M., Tian, Y., Zhu, H., and Fang, Z. (2017). Towards the automatic anime characters creation with generative adversarial networks.

[Johnson and Cook, 1983] Johnson, G. and Cook, W. (1983). A constitutive model and data for metals subjected to large strains, high strain rates and high temperatures. In *Proceedings of 7^{th} international symposium on ballistics*, The Hague, The Netherlands.

[Johnson and Holmquist, 1994] Johnson, G. R. and Holmquist, T. J. (1994). An improved computational constitutive model for brittle materials. *AIP Conference Proceedings*, 309(1):981–984.

[Kadhom, 2016] Kadhom, B. (2016). *Blast performance of reinforced concrete columns protected by FRP laminates*. PhD thesis, Université d'Ottawa/University of Ottawa.

[Karniadakis et al., 2021] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440.

[Karpatne et al., 2016] Karpatne, A., Atluri, G., Faghmous, J. H., Steinbach, M. S., Banerjee, A., Ganguly, A. R., Shekhar, S., Samatova, N. F., and Kumar, V. (2016). Theory-guided data science: A new paradigm for scientific discovery. *CoRR*, abs/1612.08544.

[Karras, 2017] Karras, T. (2017). Papers with code - celeba-hq dataset.

[Karras et al., 2018a] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018a). Progressive growing of gans for improved quality, stability, and variation.

[Karras et al., 2018b] Karras, T., Laine, S., and Aila, T. (2018b). A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948.

[Kaur et al., 2018] Kaur, B., Sharma, M., Mittal, M., Verma, A., Goyal, L. M., and Hemanth, D. J. (2018). An improved salient object detection algorithm combining background and foreground connectivity for brain image analysis. *Computers Electrical Engineering*, 71:692–703.

[Khouri, 2009] Khouri, M. (2009). Estimation of the maximum allowable drift at the top of a shear wall (within elastic limits).

[Kingma and Welling, 2014] Kingma, D. and Welling, M. (2014). Auto-encoding variational bayes. *arXiv:1312.6114*, 10:1–9.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kodali et al., 2017] Kodali, N., Abernethy, J., Hays, J., and Kira, Z. (2017). On convergence and stability of gans.

[Kowalski et al., 2020] Kowalski, M., Garbin, S. J., Estellers, V., Baltrušaitis, T., Johnson, M., and Shotton, J. (2020). Config: Controllable neural face image generation. In *European Conference on Computer Vision*, pages 299–315. Springer.

[Krauthammer, 2008] Krauthammer, T. (2008). *Modern Protective Structures*. CRC Press.

[Krishnan et al., 2010] Krishnan, K., Sockalingam, S., Bansal, S., and Rajan, S. (2010). Numerical simulation of ceramic composite armor subjected to ballistic impact. *Composites Part B: Engineering*, 41(8):583–593.

[Kristoffersen et al., 2020a] Kristoffersen, M., Costas, M., Koenis, T., Brøtan, V., Paulsen, C. O., and Børvik, T. (2020a). On the ballistic perforation resistance of additive manufactured alsi10mg aluminium plates. *International Journal of Impact Engineering*, 137:103476.

[Kristoffersen et al., 2020b] Kristoffersen, M., Costas, M., Koenis, T., Brøtand, V., Paulsen, C., and Børvik, T. (2020b). On the ballistic perforation resistance of additive manufactured AlSi10Mg t aluminium plates. *International Journal of Impact Engineering*, 137:1–8.

[Kristoffersen et al., 2021] Kristoffersen, M., Toreskås, O. L., Dey, S., and Børvik, T. (2021). Ballistic perforation resistance of thin concrete slabs impacted by ogive-nose steel projectiles. *International Journal of Impact Engineering*, 156:103957.

[Krizhevsky and Hinton, 2009] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario.

[Krizhevsky et al., 2017] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90.

[Landkof and Goldsmith, 1985] Landkof, B. and Goldsmith, W. (1985). Petalling of thin, metallic plates during penetration by cylindro-conical projectiles. *International Journal of Solids and Structures*, 21(3):245–266.

[Larsen et al., 2015] Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

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

[Ledig et al., 2017] Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network.

[Li et al., 2020] Li, H., Krcek, M., and Perin, G. (2020). *A Comparison of Weight Initializers in Deep Learning-Based Side-Channel Analysis*, pages 126–143. Springer.

[Li et al., 2016a] Li, Z., K G, P., Deng, Y., Raabe, D., and Tasan, C. (2016a). Metastable high-entropy dual-phase alloys overcome the strength-ductility trade-off. *Nature*.

[Li et al., 2016b] Li, Z., Pradeep, K., Deng, Y., Raabe, D., and Tasan, C. (2016b). Metastable high-entropy dual-phase alloys overcome the strength–ductility trade-off. *Nature*, 543:227–230.

[Liaghat et al., 2013] Liaghat, G., Shanazari, H., Tahmasebi, M., Aboutorabi, A., and Hadavinia, H. (2013). A modified analytical model for analysis of perforation of projectile into ceramic composite targets. *Int J Compos Mater*, 3(6):17–22. cited By 7.

[Liu et al., 2018] Liu, J., Long, Y., Ji, C., Liu, Q., Zhong, M., and Zhou, Y. (2018). Influence of layer number and air gap on the ballistic performance of multi-layered targets subjected to high velocity impact by copper efp. *International Journal of Impact Engineering*, 112:52–65.

[Liu, 2003] Liu, L. (2003). The optimization design on Metal/Ceramic FGM armor with neural net and conjugate gradient method. *Materials Science Forum*, 423(425):791–796.

[Liu and Shi, 2020] Liu, S. and Shi, Q. (2020). Multitask deep learning with spectral knowledge for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters*, 17(12):2110–2114.

[Liu et al., 2017] Liu, W. K., Ma, Z., Li, S., and Chang, F. K. (2017). Advanced material modeling in ls-dyna. *International Journal of Impact Engineering*, 107:41–58.

[Liu et al., 2019] Liu, Y., Yan, J., Li, Z., and Huang, F. (2019). Improved sdof and numerical approach to study the dynamic response of reinforced concrete columns subjected to close-in blast loading. *Structures*, 22:341–365.

[Lu et al., 2020] Lu, L., Karniadakis, G., Yazdani, A., and Raissi, M. (2020). Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS Computational Biology*, 16:e1007575.

[Maas, 2013] Maas, A. L. (2013). Rectifier nonlinearities improve neural network acoustic models.

[Maas et al., 2013] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

[Majchrzak et al., 2016] Majchrzak, A., Markus, M., and Wareham, J. (2016). Designing for digital transformation: lessons for information systems research from the study of ict and societal challenges. *Management Information Systems Quarterly*, 40:267–277.

[Masci et al., 2011] Masci, J., Meier, U., Cireșan, D., and Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In Honkela, T., Duch, W., Girolami, M., and Kaski, S., editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 52–59, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Masri and Ryan, 2021] Masri, R. and Ryan, S. (2021). Ballistic limit predictions for perforation of multi-layered aluminium armour targets by rigid, nose-pointed projectiles. *International Journal of Impact Engineering*, 155:103900.

[Masri et al., 2000] Masri, S. F., Smyth, A. W., Chassiakos, A. G., Caughey, T. K., and Hunter, N. F. (2000). Application of neural networks for detection of changes in nonlinear systems. *Journal of Engineering Mechanics*, 126(7):666–676.

[McCarthy et al., 2006] McCarthy, J., Minsky, M., Rochester, N., and Shannon, C. (2006). A proposal for the dartmouth summer research project on arti cial intelligence. *AI Magazine*, 27.

[Meyers et al., 2006] Meyers, M. A., Mishra, A., and Benson, D. J. (2006). *Mechanical Metallurgy*. Cambridge University Press, 2nd edition.

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *CoRR*, abs/1411.1784.

[Mohammad et al., 2020] Mohammad, Z., Gupta, P. K., and Baqi, A. (2020). Experimental and numerical investigations on the behavior of thin metallic plate targets subjected to ballistic impact. *International Journal of Impact Engineering*, 146:103717.

[Mohan and Gaitonde, 2018] Mohan, A. T. and Gaitonde, D. V. (2018). A deep learning based approach to reduced order modeling for turbulent flow control using lstm neural networks.

[Murphy, 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[Nagamani Devi and Vijayalakshmi, 2021] Nagamani Devi, G. and Vijayalakshmi, M. (2021). Smart structural health monitoring in civil engineering: A survey. *Materials Today: Proceedings*, 45:7143–7146. International Conference on Mechanical, Electronics and Computer Engineering 2020: Materials Science.

[Naik et al., 2013] Naik, N., Kumar, S., Ratnaveer, D., Joshi, M., and Akella, K. (2013). An energy-based model for ballistic impact analysis of ceramic-composite armors. *International Journal of Damage Mechanics*, 22(2):145–187.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress.

[Neal, 1992] Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113.

[Newmark, 1959] Newmark, N. M. (1959). A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division*, 85(3):67–94.

[Nowozin et al., 2016] Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-gan: Training generative neural samplers using variational divergence minimization.

[Nwankpa et al., 2018] Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning.

[O'Gorman and Dwyer, 2018] O'Gorman, P. and Dwyer, J. (2018). Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. *Journal of Advances in Modeling Earth Systems*, 10.

[Olgac and Karlik, 2011] Olgac, A. and Karlik, B. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence And Expert Systems*, 1:111–122.

[Oppenheim, 1996] Oppenheim, I. (1996). A survey of thermodynamics. M. Bailyn, AIP Press, New York, 1994. *Journal of Statistical Physics*, 83(3-4):791–792.

[Oulbacha and Kadoury, 2020] Oulbacha, R. and Kadoury, S. (2020). Mri to ct synthesis of the lumbar spine from a pseudo-3d cycle gan. In *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, pages 1784–1787.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1310–III–1318. JMLR.org.

[Pathan et al., 2019] Pathan, M. V., Ponnusami, S. A., Pathan, J., Pitisongsawat, R., Erice, B., Petrinic, N., and Tagarielli, V. L. (2019). Predictions of the mechanical properties of unidirectional fibre composites by supervised machine learning. *Sci. Rep.*, 9(1):13964.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Polyak, 1964] Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[Radford et al., 2016] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

[Rai and Sahu, 2020] Rai, R. and Sahu, C. K. (2020). Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus. *IEEE Access*, PP:1–1.

[Ransbotham et al., 2016] Ransbotham, S., Fichman, R. G., Gopal, R., and Gupta, A. (2016). Special section introduction—ubiquitous it and digital vulnerabilities. *Information Systems Research*, 27(4):834–847.

[Recht and Ipson, 1963] Recht, R. F. and Ipson, T. W. (1963). Ballistic perforation dynamics. *Journal of Applied Mechanics*, 30:384–390.

[Reddy, 2018] Reddy, J. (2018). *Introduction to the Finite Element Method 4E*. McGraw-Hill Education.

[Reed and Marks, 1998] Reed, R. D. and Marks, R. J. (1998). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, Cambridge, MA, USA.

[Rodriguez-Millan et al., 2018] Rodriguez-Millan, M., Garcia-Gonzalez, D., Rusinek, A., Abed, F., and Arias, A. (2018). Perforation mechanics of 2024 aluminium protective plates subjected to impact by different nose shapes of projectiles. *Thin-Walled Structures*, 123:1 – 10.

[Rosenberg et al., 2016] Rosenberg, Z., Kositski, R., and Dekel, E. (2016). On the perforation of aluminum plates by 7.62mm apm2 projectiles. *International Journal of Impact Engineering*, 97:79 – 86.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

[Ryan and Thaler, 2013] Ryan, S. and Thaler, S. (2013). Artificial neural networks for characterising whipple shield performance. *International Journal of Impact Engineering*, 56:61–70. Selected papers from the 2012 Hypervelocity Impact Symposium.

[Ryan et al., 2016] Ryan, S., Thaler, S., and Kandanaarachchi, S. (2016). Machine learning methods for predicting the outcome of hypervelocity impact events. *Expert Systems with Applications*, 45:23 – 39.

[Sabadin et al., 2018] Sabadin, G., Gaiotti, M., Rizzo, C. M., and Bassano, A. (2018). Development and validation of a numerical model for the simulation of high-velocity impacts on advanced composite armor systems. *Nonlinear Dynamics*, 91(3):1791–1816.

[Saeedi and Giusti, 2023] Saeedi, J. and Giusti, A. (2023). Semi-supervised visual anomaly detection based on convolutional autoencoder and transfer learning. *Machine Learning with Applications*, 11:100451.

[Sainath et al., 2015] Sainath, T. N., Kingsbury, B., Saon, G., Soltau, H., rahman Mohamed, A., Dahl, G., and Ramabhadran, B. (2015). Deep convolutional neural networks for large-scale speech tasks. *Neural Networks*, 64:39–48. Special Issue on "Deep Learning of Representations".

[Salimans et al., 2016] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans.

[Scazzosi et al., 2021] Scazzosi, R., Giglio, M., and Manes, A. (2021). Experimental and numerical investigation on the perforation resistance of double-layered metal shields under high-velocity impact of soft-core projectiles. *Engineering Structures*, 228:111467.

[Schellbach, 1851] Schellbach, K. (1851). Probleme der variationsrechnung. 1851(41):293–363.

[Schleder et al., 2019] Schleder, G. R., Padilha, A. C. M., Acosta, C. M., Costa, M., and Fazzio, A. (2019). From DFT to machine learning: recent approaches to materials science–a review. *Journal of Physics: Materials*, 2(3):032001.

[Schlegl et al., 2017] Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery.

[Schwartz et al., 2019] Schwartz, L. C. M. W., Ellekilde, L.-P., and Krüger, N. (2019). Automated fixture design using an imprint-based design approach amp; optimisation in simulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 54–61.

[Shaziya, 2020] Shaziya, H. (2020). A study of the optimization algorithms in deep learning.

[Sherstinsky, 2020] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

[Shin et al., 2018] Shin, H.-C., Tenenholtz, N. A., Rogers, J. K., Schwarz, C. G., Senjem, M. L., Gunter, J. L., Andriole, K., and Michalski, M. (2018). Medical image synthesis for data augmentation and anonymization using generative adversarial networks.

[Shixin et al., 2022] Shixin, P., Kai, C., Tian, T., and Jingying, C. (2022). An autoencoder-based feature level fusion for speech emotion recognition. *Digital Communications and Networks*.

[Sikarwar et al., 2014] Sikarwar, R. S., Velmurugan, R., and Gupta, N. (2014). Influence of fiber orientation and thickness on the response of glass/epoxy composites subjected to impact loading. *Composites Part B: Engineering*, 60:627 – 636.

[Sixt et al., 2018] Sixt, L., Wild, B., and Landgraf, T. (2018). Rendergan: Generating realistic labeled data. *Frontiers in Robotics and AI*, 5:66.

[Smith, 2015] Smith, L. N. (2015). No more pesky learning rate guessing games. *CoRR*, abs/1506.01186.

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.

[Taylor, 1948] Taylor, G. I. (1948). The formation and enlargement of a circular hole in a thin plastic sheet. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):103–124.

[Teixeira-Dias et al., 2020] Teixeira-Dias, F., Smith, N., and Galiounas, E. (2020). *Analytical and Energy-Based Methods for Penetration Mechanics*, pages 50–61. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Teixeira-Dias et al., 2019] Teixeira-Dias, F., Thompson, S., and Paulino, M. (2019). *An Artificial Intelligence-based Hybrid Method for Multi-layered Armour Systems*, pages 381–400. Springer International Publishing, Cham.

[Theis et al., 2015] Theis, L., van den Oord, A., and Bethge, M. (2015). A note on the evaluation of generative models.

[Thompson et al., 2021] Thompson, S., Teixeira-Dias, F., Paulino, M., and Hamilton, A. (2021). Ballistic response of armour plates using generative adversarial networks. *Defence Technology*.

[Thompson et al., 2022] Thompson, S., Teixeira-Dias, F., Paulino, M., and Hamilton, A. (2022). Predictions on multi-class terminal ballistics datasets using conditional generative adversarial networks. *Neural Networks*, 154:425–440.

[Thomson, 1955] Thomson, W. T. (1955). An approximate theory of armor penetration. *Journal of Applied Physics*, 26(1):80–82.

[Tria and Trebinski, 2015] Tria, D. E. and Trebinski, R. (2015). On the influence of fracture criterion on perforation of high-strength steel plates subjected to armour piercing projectile / wpływ kryterium pekania materiału na perforacje płyt ze stali pancernej pociskiem przeciwpancernym. *Archive of Mechanical Engineering*, 62.

[Turing, 1950] Turing, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460.

[Uçar et al., 2017] Uçar, A., Demir, Y., and Güzeliş, C. (2017). Object recognition and detection with deep learning for autonomous driving applications. *SIMULATION*, 93(9):759–769.

[Vemuri and Bhat, 2011] Vemuri, M. and Bhat, T. (2011). Armour protection and affordable protection for futuristic combat vehicles. *Defence Science Journal*, 61.

[von Krogh, 2018] von Krogh, G. (2018). Artificial intelligence in organizations: New opportunities for phenomenon-based theorizing. *Academy of Management Discoveries*, 4(4):404–409.

[Wang et al., 2016] Wang, D., Khosla, A., Gargeya, R., Irshad, H., and Beck, A. H. (2016). Deep learning for identifying metastatic breast cancer.

[Wei et al., 2012] Wei, Z., Yunfei, D., Sheng, C. Z., and Gang, W. (2012). Experimental investigation on the ballistic performance of monolithic and layered metal plates subjected to impact by blunt rigid projectiles. *International Journal of Impact Engineering*, 49:115 – 129.

[Wiley, 2005] Wiley, J. (2005). *Information Theory and Statistics*, chapter 11, pages 347–408. John Wiley Sons, Ltd.

[Willard et al., 2020] Willard, J., Jia, X., Xu, S., Steinbach, M., and Kumar, V. (2020). Integrating physics-based modeling with machine learning: A survey. *ArXiv*, abs/2003.04919.

[Woodward, 1987] Woodward, R. L. (1987). A structural model for thin plate perforation by normal impact of blunt projectiles. *International Journal of Impact Engineering*, 6(2):129–140.

[Wu and Jahanshahi, 2019] Wu, R.-T. and Jahanshahi, M. R. (2019). Deep convolutional neural network for structural dynamic response estimation and system identification. *Journal of Engineering Mechanics*, 145(1):04018125.

[Wu et al., 2016a] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016a). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

[Wu et al., 2016b] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, , Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016b). Google's neural machine translation system: Bridging the gap between human and machine translation.

[Xiao et al., 2019a] Xiao, X., Pan, H., Bai, Y., Lou, Y., and Chen, L. (2019a). Application of the modified mohr–coulomb fracture criterion in predicting the ballistic resistance of 2024-t351 aluminum alloy plates impacted by blunt projectiles. *International Journal of Impact Engineering*, 123:26 – 37.

[Xiao et al., 2019b] Xiao, X., Wang, Y., Vershinin, V. V., Chen, L., and Lou, Y. (2019b). Effect of lode angle in predicting the ballistic resistance of weldox 700 e steel plates struck by blunt projectiles. *International Journal of Impact Engineering*, 128:46 – 71.

[Xie et al., 2020] Xie, Q., Dai, Z., Hovy, E., Luong, T., and Le, Q. (2020). Unsupervised data augmentation for consistency training. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6256–6268. Curran Associates, Inc.

[Xu et al., 2015] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network.

[Xu et al., 2019] Xu, L., Xu, H., and Wen, H. (2019). On the penetration and perforation of concrete targets struck transversely by ogival-nosed projectiles - a numerical study. *International Journal of Impact Engineering*, 125:39–55.

[Yang et al., 2022] Yang, A., Romanyk, D., and Hogan, J. D. (2022). High-velocity impact study of an advanced ceramic using finite element model coupling with a machine learning approach. *Ceramics International*.

[Yang et al., 2019] Yang, D., Xiong, T., Xu, D., and Kevin Zhou, S. (2019). *Segmentation using adversarial image-to-image networks*. Cited by: 6.

[Ye et al., 2020] Ye, G., Zhang, Z., Ding, L., Li, Y., and Zhu, Y. (2020). Gan-based focusing-enhancement method for monochromatic synthetic aperture imaging. *IEEE Sensors Journal*, 20(19):11484–11489.

[Yin et al., 2017a] Yin, W., Kann, K., Yu, M., and Schütze, H. (2017a). Comparative study of cnn and rnn for natural language processing.

[Yin et al., 2017b] Yin, W., Kann, K., Yu, M., and Schütze, H. (2017b). Comparative study of cnn and rnn for natural language processing.

[Yu et al., 2020] Yu, Y., Huang, Z., Li, F., Zhang, H., and Le, X. (2020). Point encoder gan: A deep learning model for 3d point cloud inpainting. *Neurocomputing*, 384:192–199.

[Yunfei et al., 2014a] Yunfei, D., Wei, Z., Guanghui, Q., Gang, W., Yonggang, Y., and Peng, H. (2014a). The ballistic performance of metal plates subjected to impact by blunt-nosed projectiles of different strength. *Materials Design (1980-2015)*, 54:1056–1067.

[Yunfei et al., 2014b] Yunfei, D., Wei, Z., Yonggang, Y., Lizhong, S., and gang, W. (2014b). Experimental investigation on the ballistic performance of double-layered plates subjected to impact by projectile of high strength. *International Journal of Impact Engineering*, 70:38–49.

[Zhang et al., 2017] Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., and Metaxas, D. (2017). Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks.

[Zhang et al., 2020] Zhang, S., Wang, L., Chang, C., Liu, C., Zhang, L., and Cui, H. (2020). An image denoising method based on bm4d and gan in 3d shearlet domain. *Mathematical Problems in Engineering*, 2020.

[Zhang and Stronge, 1996] Zhang, T. and Stronge, W. (1996). Theory for ballistic limit of thin ductile tubes hit by blunt missiles. *International Journal of Impact Engineering*, 18(7):735 – 752.

[Zhou and Stronge, 2008] Zhou, D. and Stronge, W. (2008). Ballistic limit for oblique impact of thin sandwich panels and spaced plates. *International Journal of Impact Engineering*, 35(11):1339 – 1354.

[Zimmermann et al., 2019] Zimmermann, C., Ceylan, D., Yang, J., Russell, B., Argus, M., and Brox, T. (2019). Freihand: A dataset for markerless capture of hand pose and shape from single rgb images.

[Zukas, 1980] Zukas, J. A. (1980). Impact dynamics: Theory and experiment.

[Zukas, 1992] Zukas, J. A. (1992). *Impact dynamics*. Krieger Pub. Co.

[Zukas et al., 1983] Zukas, J. A., Nicholas, T., Swift, H. F., Greszczuk, L. B., Curran, D. R., and Malvern, L. E. (1983). Impact Dynamics. *Journal of Applied Mechanics*, 50(3):702.