School of Computing and Information Systems, The University of Melbourne

Machine Learning with Adversarial Perturbations and Noisy Labels

Xingjun Ma

Submitted in total fulfilment of the requirements of the degree of Doctor of Philosophy

Produced on archival quality paper

August 2018
Abstract

Machine learning models such as traditional random forests (RFs) and modern deep neural networks (DNNs) have been successfully used to solve complex learning problems in many applications such as speech recognition, image classification, face recognition, gaming agents and self-driving cars. For example, DNNs have demonstrated near or even surpassing human-level performance in image classification tasks. Despite their current success, these models are still vulnerable to noisy real-world situations where illegitimate or noisy data may exist to corrupt learning. Studies have shown that by adding small, human imperceptible (in the case of images) adversarial perturbations, normal samples can be perturbed into “adversarial examples”, and DNNs can be made to misclassify adversarial examples with a high level of confidence. This arouses security concerns when employing DNNs in security-sensitive applications such as fingerprint recognition, face verification and autonomous cars. Studies have also found that DNNs can overfit to noisy (incorrect) labels and as a result, generalize poorly. This has been one of the key challenges when applying DNNs in noisy real-world scenarios where even high-quality datasets tend to contain noisy labels. Another open question in machine learning is whether actionable knowledge (or feedback) can be generated from prediction models to support decision making towards some long-term learning goals (for example, mastering a certain type of skills in a simulation-based learning (SBL) environment). We view the feedback generation problem from a new perspective of adversarial perturbation, and explore the possibility of using adversarial techniques to generate feedback.

In this thesis, we investigate machine learning models including DNNs and RFs, and their learning behavior through the lens of adversarial perturbations
and noisy labels, with the aim of achieving more secure and robust machine learning. We also explore the possibility of using adversarial techniques in a real-world application: to support skill acquisition in SBL environments through the provision of performance feedback.

The first part of our work is on the investigation of DNNs and their vulnerability to adversarial perturbations, in the context of image classification. In contrast to existing work, we develop new understandings of adversarial perturbations by exploring DNN representation space with the Local Intrinsic Dimensionality (LID) measure. In particular, we characterize adversarial subspaces in the vicinity of adversarial examples using LID, and find that adversarial subspaces are of higher intrinsic dimensionality than normal data subspaces. We not only provide a theoretical explanation of the high dimensionality of adversarial subspaces, but also empirically demonstrate that such properties can be used to effectively discriminate adversarial examples generated using state-of-the-art attacking methods.

The second part of our work is to explore the possibility of using adversarial techniques in a beneficial way to generate interactive feedback for intelligent tutoring in SBL environments. Feedback is actions (in the form of feature changes) generated from a pre-trained prediction model that can be delivered to a learner in an SBL environment to correct mistakes or improve skills. We demonstrate that such feedback can be generated accurately and efficiently using properly constrained adversarial techniques with DNNs.

In addition to DNNs, we also explore, in the third part of our work, adversarial feedback generation from RF models. Adversarial perturbations can be easily generated from DNNs using gradient descent and backpropagation, however, it is still an open question whether such perturbations can be generated from models such as RFs that do not work with gradients. This part of our work confirms that adversarial perturbations can also be crafted from RFs for the provision of feedback in SBL. In particular, we propose a perturbation method that can find the optimal space transition from one undesired class (e.g. ‘novice’) to the desired class (e.g. ‘expert’), based on a geometric view of the RF decision space as overlapping high dimensional rectangles. We demonstrate empirically that our proposed method has high effectiveness as well as high ef-
ficiency when compared to existing methods, making it possible to be used for real-time feedback generation in SBL.

The fourth part of our work focuses on DNNs and noisy label learning: training accurate DNNs on data with noisy labels. In this work, we investigate the learning behaviours of DNNs, and show that DNNs exhibit two distinct learning styles when trained on clean versus noisy labels. A LID-based characterization of the intrinsic dimensionality of DNN subspace (inspired by the first part of our work) allows us to identify the two stages of learning from dimensionality compression to dimensionality expansion on datasets with noisy labels. Based on the observation that dimensionality expansion is associated with overfitting to noisy labels, we further propose a heuristic learning strategy to avoid the later stage of dimensionality expansion, so as to robustly train DNNs in the presence of noisy labels.

In summary, this work has contributed significantly to existing knowledge through: novel dimensional characterization of DNNs, effective discrimination of adversarial attacks, robust deep learning strategies against noisy labels, and novel approaches to feedback generation. All work is supported by theoretical analysis, empirical results and publications.
Declaration

This is to certify that

1. the thesis comprises only my original work towards the degree of Doctor of Philosophy except where indicated in the Preface,

2. due acknowledgment has been made in the text to all other material used,

3. the thesis is fewer than 80,000 words in length, exclusive of tables, maps, bibliographies and appendices.

________________________________________

Xingjun Ma
Preface

This thesis has been written at the School of Computing and Information Systems, The University of Melbourne. The major parts of the thesis are the Chapters 3, 4, 5 and 6. They are based on published proceeding papers and I declare that I am the primary author and have contributed > 50% in all of the following papers.


Acknowledgments

Without the supports of my supervisors, my colleagues, my family and my friends, it could not be possible for me to finish this thesis and this wonderful journey of PhD. My PhD study at the University of Melbourne with Prof. James Bailey and Dr. Sudanthi Wijewickrema has been nothing short of amazing. I have been given unique opportunities to work on challenging machine learning problems, present at top machine learning conferences, collaborate with researchers both inside and outside of the university, and intern at research institutes such as National Institute of Informatics (NII), Japan. These research experiences have given me a thorough grounding in research techniques, and prepared me for a life of research. Here I would like to acknowledge and thank those who supported me during this fruitful journey.

First and foremost I would like to express my deepest appreciation to my supervisors Prof. James Bailey and Dr. Sudanthi Wijewickrema. They have been giving me constant supports throughout my entire PhD. This research not only requires theoretical understanding of different machine learning models but also involves development of novel techniques for real-world applications such as virtual reality and simulated surgeries. James and Sudanthi have provided me valuable suggestions on almost every aspect of this research, and more importantly, maximum freedom to explore interesting ideas. I am also very grateful for their encouragement and support when I was struggling with the research during the first year of my PhD. Without their guidance and support, this thesis wouldn’t be possible. I would also like to thank my committee chair Prof. Reeva Lederman for good advice on my PhD career.

I would also like to express my appreciation to Dr. Yun Zhou for helping me start my research smoothly. He led me into the research on feedback generation
in virtual reality based surgical training, and shared a lot of previous work and research experiences with me at the beginning of my PhD. I would also like to thank Prof. Stephen O’Leary for teaching me surgical knowledge and the associated surgical training techniques from a surgeon point of view, and offering me the opportunity to work as a research assistant on the feedback project at the Royal Victorian Eye and Ear Hospital.

My research involves several collaborations with researchers inside and outside of the university. Here I would like to thank them for their contributions and supports in finishing this thesis. Many thanks to Prof. Michael E. Houle from NII for his valuable inputs on dimensionality theory of deep networks, and for hosting me as an intern at NII to continue my research on Local Intrinsic Dimensionality and its theory. Meanwhile, I would like to thank Dr. Yisen Wang from Tsinghua University and Dr. Bo Li from University of California, Berkeley for many insightful discussions on deep adversarial learning and noisy label learning. I also want to thank Dr. Sarah M. Erfani for helpful advice on adversarial detection.

Meanwhile, I would like to say thank you to my colleagues in the lab: Shuo, Florin, Yunzhe, Jiazhen, Simone, Yamuna, Yang, Donia and Sobia. Thank you very much for those wonderful moments, you have made my PhD journey more colorful.

Finally, I would like to sincerely thank my parents Wenhua and Yuanyou, my brother Xingfa and my sister-in-law Hui for supporting and encouraging me during this PhD journey. Finishing this PhD journey wouldn’t be possible without the support of my wife Ziyi and my lovely daughter Qiaoxi, thank them sincerely for their enormous ongoing supports.
Contents

1 Introduction 3
  1.1 Motivations of this research . . . . . . . . . . . . . . . . . . . . 4
  1.2 Research contributions . . . . . . . . . . . . . . . . . . . . . . . 7

2 Background 11
  2.1 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
    2.1.1 Supervised machine learning . . . . . . . . . . . . . . . . . 11
    2.1.2 Deep neural networks . . . . . . . . . . . . . . . . . . . . . . 13
      2.1.2.1 Feedforward neural networks . . . . . . . . . . . 15
      2.1.2.2 Convolutional neural networks . . . . . . . . . . 20
    2.1.3 Random forests . . . . . . . . . . . . . . . . . . . . . . . . . 24
  2.2 Adversarial perturbations and deep neural networks . . . . . . . 26
    2.2.1 Adversarial perturbations . . . . . . . . . . . . . . . . . . . 26
    2.2.2 Adversarial attack . . . . . . . . . . . . . . . . . . . . . . . 29
    2.2.3 Adversarial defense . . . . . . . . . . . . . . . . . . . . . . . 33
    2.2.4 Adversarial detection . . . . . . . . . . . . . . . . . . . . . . 34
  2.3 Feedback generation and simulation-based learning . . . . . . . 37
    2.3.1 Feedback generation . . . . . . . . . . . . . . . . . . . . . . 37
    2.3.2 Simulation-based learning . . . . . . . . . . . . . . . . . . . 40
    2.3.3 Feedback generation in simulation-based learning . . . . . . 42
  2.4 Noisy labels and deep neural networks . . . . . . . . . . . . . . 47
    2.4.1 Noisy labels . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
    2.4.2 Traditional learning with noisy labels . . . . . . . . . . . . 50
    2.4.3 Deep learning with noisy labels . . . . . . . . . . . . . . . . 52
## 3 Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality

3.1 Introduction ............................................. 56

3.2 Local Intrinsic Dimensionality ......................... 58

3.2.1 Definition of LID ..................................... 59

3.2.2 Estimation of LID ..................................... 60

3.3 Characterizing adversarial regions ....................... 61

3.3.1 LID of adversarial subspaces ....................... 61

3.3.2 Efficiency through minibatch sampling .............. 62

3.3.3 Using LID to characterize adversarial examples .... 63

3.4 Evaluating LID-based characterization of adversarial examples .......... 65

3.4.1 Experimental setup ................................... 65

3.4.1.1 Training and testing setup ....................... 65

3.4.1.2 Deep neural networks for pretraining ............ 66

3.4.1.3 Parameter tuning .................................. 67

3.4.2 LID characteristics of adversarial examples .......... 68

3.4.3 Analysis of LID properties ........................... 70

3.4.3.1 LID outperforms KD and BU ...................... 70

3.4.3.2 Generalizability analysis ........................ 71

3.4.3.3 Effect of larger minibatch sizes in LID estimation . 72

3.4.3.4 Adaptive attack against LID measurement ....... 73

3.5 Chapter summary ...................................... 75

## 4 Adversarial Feedback Generation with Neural Networks

4.1 Introduction ............................................. 78

4.2 Feedback generation problem .......................... 79

4.3 Proposed method ...................................... 79

4.4 Experimental validation ................................ 82

4.4.1 Datasets ............................................. 83

4.4.2 Compared methods .................................. 84

4.4.3 Experimental setup .................................. 84

4.4.4 Parameter tuning .................................... 85

4.4.5 Results ............................................ 86
5 Adversarial Feedback Generation with Random Forests

5.1 Introduction

5.2 Random forest based feedback generation

5.2.1 A geometric view

5.2.2 Random forest discretization

5.2.3 Hyper-rectangle pruning

5.2.4 Discrete approximation

5.3 Experimental evaluation

5.3.1 Experimental settings

5.3.2 Results

5.3.3 Parameter and scalability analysis

5.4 Chapter summary

6 Dimensionality-Driven Learning with Noisy Labels

6.1 Introduction

6.2 Dimensionality of deep representation subspaces

6.2.1 Local Intrinsic Dimensionality and its estimation

6.2.2 Subspace dimensionality and noisy labels

6.2.3 Two-Stage of learning of DNNs on noisy labels

6.3 Dimensionality-Driven learning strategy

6.4 Experiments

6.4.1 Empirical understanding of D2L

6.4.1.1 Experimental setup

6.4.1.2 Competing strategies

6.4.1.3 Effect on subspace learning

6.4.1.4 Effect on hypothesis learning

6.4.1.5 Effect on representation learning

6.4.1.6 Parameter sensitivity

6.4.2 Robustness against noisy labels

6.4.2.1 Experimental setup

6.4.2.2 Results
Chapter summary .................................................. 121

Conclusion and Future Works .................................... 123
  7.1 Summary of contributions ................................... 123
  7.2 Thesis limitations and future work ....................... 125
# List of Figures

2.1 A 2-layer artificial neural network. .............................................. 14  
2.2 Neural activation functions. ...................................................... 16  
2.3 Convolutional neural network architecture. ................................. 21  
2.4 A 2D illustration of the convolutional operation. .......................... 21  
2.5 Example of a CNN. .................................................................. 23  
2.6 Random forests. ....................................................................... 24  
2.7 Example of a random forest. ....................................................... 25  
2.8 An example of adversarial perturbations. ...................................... 27  
2.9 Adversarial MNIST handwritten digits. ........................................ 28  
2.10 Virtual reality based surgery simulation. ...................................... 41  
2.11 Feedback process in simulation-based learning. ............................ 44  
2.12 An example of delivering feedback in text and audio. .................. 44  
2.13 Types of noisy labels. ................................................................ 50  
3.1 Dimensional characterization of adversarial subspace ...................... 58  
3.2 High dimensionality of CIFAR-10 adversarial examples. ............... 69  
3.3 High dimensionality of MNIST adversarial examples. .................... 69  
3.4 Parameter tuning of KD and LID. ................................................. 70  
3.5 Larger batch estimation of LID. .................................................. 73  
4.1 Virtual reality surgical simulator. .................................................. 83  
4.2 Effect of simplicity regularization by $\lambda$. ................................. 86  
4.3 Effectiveness and time-cost evaluation of NNFB. ............................ 88  
5.1 An example of a random forest. ................................................... 94  
5.2 An 2D example of discrete approximation. ................................. 95
List of Tables

2.1 Summary of existing feedback generation methods . . . . . . . . 46
2.2 Types of labels for a ‘jasmine-cat’ dataset. . . . . . . . . . . . . . 49
3.1 Statistics of adversarial attacks. . . . . . . . . . . . . . . . . . . . 67
3.2 Discrimination power comparison among LID, KD and BU . . . . 71
3.3 Generalizability of LID-based discrimination. . . . . . . . . . . . . 72
3.4 The failure rates of an adaptive attack. . . . . . . . . . . . . . . . . 74
4.1 Effectiveness evaluation by 6 different classifiers on dataset \( D_1 \). . 87
4.2 Effectiveness evaluation by 6 different classifiers on dataset \( D_2 \). . 87
4.3 Time-cost evaluation on datasets \( D_1 \) and \( D_2 \). . . . . . . . . . . . . 88
5.1 Performance comparison between DA and other methods. . . . . 101
6.1 Robustness of D2L on MNIST with varying noisy label rates . . . . 120
6.2 Robustness of D2L on SVHN with varying noisy label rates. . . . . 121
6.3 Robustness of D2L on CIFAR-10 with varying noisy label rates. . . 121
6.4 Robustness of D2L on CIFAR-100 with varying noisy label rates. . . 122
Chapter 1

Introduction

Although machine learning models such as traditional random forests (RFs) and modern deep neural networks (DNNs) have demonstrated superior performance on many real-world problems (for example, speech recognition [74], image classification [101], face recognition [155], gaming agents [185] and autonomous cars [25]), the working mechanisms of these models are still not completely understood. This has led to several challenges when applying these models to noisy real-world scenarios. First, studies have shown that DNNs are vulnerable to adversarial perturbations [202]. Adversarial perturbations are test-time input noise generated by an attacking method (or adversary) manipulating a pre-trained DNN model, and the perturbed samples are known as “adversarial examples”. Adversarial perturbations are often small, imperceptible to human observers in the context of images, but are able to fool DNNs to make misclassifications with high confidence. This has become a major security concern when deploying DNNs in security-sensitive applications such as face verification and autonomous cars [45, 63].

Second, although DNNs trained on clean data generalize well to test data, they perform poorly when trained on noisy labels (i.e. a certain proportion of the training samples were labelled incorrectly [157]). Noisy labels are common in real-world applications where data collection and labelling are often costly and error-prone, and as such, even high-quality datasets are likely to contain noisy labels [213]. Training accurate DNNs in the presence of noisy labels is
important yet challenging. This is because 1) it is difficult to determine whether a learning process is noisy\cite{233}, and 2) robust training strategies are needed to train accurate DNNs against noisy labels. Deep learning with noisy labels requires a comprehensive understanding of DNNs and their learning behavior with noisy labels, so as to develop robust learning strategies to avoid overfitting to noisy labels \cite{4, 102, 184}.

Third, extracting actionable knowledge from prediction models to support decision making is a challenging problem, especially in the face of noisy data. Although good prediction performance can be easily achieved by learning from labeled data, it is still an open question as to whether actionable knowledge (or feedback\cite{31}) can be generated from these models to support decision making towards some long-term learning goals (for example, mastering a certain type of skills in a simulation-based learning (SBL) environment\cite{31}). This is the case not only for deep and complex DNNs, but also for traditional machine learning models such as RFs. Feedback is actionable instructions (in the form of feature changes) that can be generated from a prediction model based on a learner’s previous behavior, and then delivered to the learner so that mistakes or poor skills can be avoided or improved. Generating feedback from prediction models also requires a deep understanding of the model’s working mechanism\cite{238}.

1.1 Motivations of this research

The above mentioned challenges in supervised learning with prediction models are closely related to the concept of “perturbations”, either feature perturbations or label perturbations. Adversarial perturbations and feedback are feature (or input) perturbations\cite{202} while noisy labels are label perturbations\cite{140}. Feedback is a type of feature perturbations\cite{31} as it consists of feature changes that can move a sample from one undesired class (e.g., ‘novice’) to a desired class (e.g., ‘expert’). To some extent, adversarial perturbation and noisy label based learning both study the learning behaviour of DNNs under exceptional data either features or labels. However, the two problems are fundamentally different: adversarial perturbations are test-time perturbations while noisy labels are
1.1. MOTIVATIONS OF THIS RESEARCH

training-time perturbations. Adversarial perturbation based learning studies test-time behaviours of a pre-trained model (trained on clean data) on perturbed test inputs. Noisy label based learning aims to learn accurate models when training labels are noisy. In contrast to both adversarial perturbations and noisy labels which are both harmful perturbations that can corrupt prediction models, feedback falls in to the category of beneficial perturbations and can be used to support decision making and interactive learning.

In this thesis, we aim to shed new light on the theoretical underpinnings of DNNs. To this end, we investigate DNNs and their learning behavior through the lens of both adversarial perturbations and noisy labels. Then, based on this new understanding of DNN behaviour, we propose 1) an effective characterization measure for discriminating adversarial examples from normal samples towards more secure deep learning, and 2) novel learning strategies for training accurate DNNs in the presence of noisy labels towards more robust deep learning. Moreover, we explore the novel use of adversarial techniques in a beneficial way, namely for feedback generation in SBL environments.

Although many works have been proposed to explain the vulnerability of DNNs to adversarial perturbations [63, 145, 202, 205], develop attacking methods to attack DNNs [6, 21, 22, 63, 124, 129, 138, 153], or develop defenses against adversarial attacks [47, 63, 66, 129, 154, 228], a comprehensive understanding of the adversarial phenomenon is still lacking. This has led to the inability to develop effective defense methods against such attacks. Exploring DNN behavior under adversarial perturbations requires a deeper understanding of DNN representation space. It is intriguing that DNNs are capable of learning complex data manifolds [74], yet small perturbations can make samples escape from normal data subspaces to adversarial subspaces where the class distribution is different. In contrast to existing work, in Chapter 3, we characterize the dimensional properties of DNN subspaces so as to understand the difference between adversarial subspaces and normal data subspaces, and how adversarial perturbations escape to adversarial subspaces. Particularly, we characterize the intrinsic dimensionality of the local regions (the so-called “adversarial subspaces” [137, 207]) of adversarial examples. Based on the observation that adversarial subspaces are of higher intrinsic dimensionality, we propose to use such dimen-
sional characteristics to effectively detect adversarial examples generated using state-of-the-art attacking methods.

Meanwhile, in contrast to existing work exploiting adversarial techniques to attack prediction models, this thesis also explores the beneficial use of adversarial techniques in feedback generation for decision making and intelligent tutoring in simulation-based learning (SBL) environments. Both adversarial perturbations and feedback are feature perturbations that can change the class of a sample. We observe this similarity between the two tasks, and explore the transfer of techniques from adversarial learning to feedback generation. More specifically, we investigate novel adversarial feedback generation methods with DNNs as well as RFs, based on geometric understandings of their decision space. Considering a binary classification problem in a SBL environment to classify a learner’s skill level to either ‘expert’ or ‘novice’, a prediction model trained to perform such a task segments the decision space to either expert or novice subspaces. Given a novice behavior, the feedback generation problem is to find the optimal transition of the behavior from its novice subspace to an expert subspace. We propose to find such transitions by adversarial techniques in the decision space of DNNs in Chapter 4 and in that of RFs in Chapter 5.

This thesis also studies the challenge of deep learning with noisy labels. Training accurate DNNs in the presence of noisy labels requires a deeper understanding of DNN learning behaviors on clean versus noisy labels [4, 233]. In contrast to existing approaches [27, 90, 104, 140, 157, 163, 193, 208, 213, 234], in Chapter 6, we tackle this challenge by first investigating the intrinsic dimensionality of DNN subspace towards a better understanding of DNN learning behavior. This is an important first step for noisy label based learning because if learning with clean versus noisy labels can be differentiated, then proper adjustments can be applied to avoid noisy learning and overfitting to noisy labels [233]. We show, from the perspective of subspace dimensionality, that DNNs indeed exhibit distinctive learning styles when trained on clean versus noisy labels, and further propose a heuristic training strategy to address the noisy label learning problem.
1.2 Research contributions

In Chapter 3, we study the dimensional properties of adversarial subspaces in the vicinity of adversarial examples generated using state-of-the-art attack methods. The contributions are as follows:

- We propose Local Intrinsic Dimensionality (LID) for the characterization of adversarial regions of deep networks. We discuss how adversarial perturbation can affect the LID characteristics of an adversarial region, and empirically show that the characteristics of test samples can be estimated effectively using a minibatch of training data.

- Our study reveals that the estimated LID of adversarial examples is significantly higher than that of normal data examples, and that this difference becomes more pronounced in deeper layers of a DNN.

- We empirically demonstrate that the LID characteristics of adversarial examples generated using five state-of-the-art attack methods can be easily discriminated from those of normal examples. We also develop a baseline classifier with features based on LID estimates that generally outperforms several existing detection measures on five attacks across three benchmark datasets. Though the adversarial examples considered here are not guaranteed to be the strongest with careful parameter tuning, these preliminary results firmly demonstrate the usefulness of the LID measure. This opens up new directions for the use of LID characteristics in adversarial defense and understanding.

- We show that the adversarial regions generated by different attacks share similar dimensional properties, in that LID characteristics of a simple attack can potentially be used to detect other more complex attacks. We also show that a naive LID-based detector is robust to the normal low confidence Optimization-based attack of Carlini et al. [21].

In Chapter 4, we explore the possibility of using adversarial techniques in a beneficial way in a real-world application. We make the following contributions:
1. INTRODUCTION

- We demonstrate how the adversarial technique can be used to generate actionable knowledge (or feedback) with neural networks, to support skill acquisition in virtual reality based surgical training.

- We propose a novel neural network based feedback generation method that works with a $L_1$ regularized loss to control the simplicity of feedback and a bounded update to ensure the generated feedback has practical meaning.

- We show that the proposed method has high effectiveness as well as high efficiency when compared to existing methods, making it possible to be used for real-time feedback generation in SBL.

We further investigate, in Chapter 5, the adversarial generation of feedback with traditional machine learning models, specifically, RFs. Here, we make the following contributions:

- We discuss the RF-based feedback formulation problem from a computational geometric point of view, for the purpose of feedback generation in virtual reality surgical training.

- We propose a novel method to formulate feedback using a RF, based on the discretization of the RF decision space.

- We demonstrate that it has near-optimal effectiveness, is highly efficient, and is scalable, make it ideal for the target application.

In Chapter 6, we investigate the learning behaviors of DNNs when the training labels are noisy and make the following contributions:

- We study the dimensionality of the deep representation subspaces learned by a DNN, and provide a dimensionality-driven explanation of DNN generalization behavior in the presence of label (class) noise. We show that from a dimensionality perspective, DNNs exhibit distinctive learning styles with clean labels versus noisy labels.
1.2. RESEARCH CONTRIBUTIONS

- We show that the local intrinsic dimensionality can be used to identify the stage shift from dimensionality compression to dimensionality expansion.

- We propose the D2L learning strategy that modifies the loss function once the turning point between the two stages of dimensionality compression and expansion is recognized, in an effort to prevent overfitting.

- We empirically demonstrate on MNIST, SVHN, CIFAR-10 and CIFAR-100 datasets that our proposed D2L learning strategy can effectively learn 1) low-dimensional representation subspaces that capture the underlying data distribution, 2) simpler hypotheses, and 3) high-quality deep representations.
Chapter 2

Background

In this chapter, we discuss the background of our research, review current literature, and identify some of the limitations and gaps in the literature that we will address in the proceeding chapters. We start with an introduction of supervised machine learning and two supervised machine learning models exploited in this thesis, i.e. deep neural networks and random forests. We then review the concept of adversarial perturbations and adversarial examples, followed by a review of existing adversarial attacking methods and defense strategies. We also discuss the feedback generation problem in SBL, and limitations of existing feedback generation approaches which we will address using adversarial techniques. Finally, we discuss deep learning with noisy labels, existing works and their limitations.

2.1 Preliminaries

In this section, we introduce the basic concepts in supervised machine learning and two investigated supervised learning models: deep neural networks and random forests.

2.1.1 Supervised machine learning

In the past decade, artificial intelligence has undergone a significant development with near human-level performances on a wide range of complex learning
problems, such as speech recognition [74], image classification [101], face recognition [155], gaming agents [185] and self-driving cars [25]. Lying in the core of artificial intelligence, machine learning techniques have been developed to formulate complex real-world problems into machine learnable tasks, and to construct models that can learn from and make predictions on data. Based on the types of data available for learning, machine learning tasks fall into three categories: supervised learning, unsupervised learning and semi-supervised learning. If training data not only contains features describing the problem but also contains labels standardizing the output solutions, then it is called labeled data, and learning from labeled data is called supervised learning. Unsupervised learning deals with unlabeled data (e.g. data that only contains features), while semi-supervised learning works with partially labeled data (e.g. a mix of both labeled and unlabeled data). Our works in this thesis fall into the scope of supervised learning.

Depends on the forms of labels available in the data, supervised learning problems can be further divided into classification problems and regression problems. Classification problems refer to learning with discrete labels, while regression problems refer to learning with continuous labels. Discrete labels are often symbolic indicators, as such they do not have numerical meaning and are not directly comparable. An example of discrete labels is the integers indicating the type of objects contained in an image, for instance, 0 represents ‘cat’ and 1 represents ‘dog’. Continuous labels are real numbers with numerical meaning and are directly comparable to each other. Such examples include stock/house price, weather temperature and traffic volume at a certain place/time, to name a few. In this thesis, we investigate machine learning models in the context of classification problems.

Given a dataset \((X, Y) = \{(x, y)|x \in X, y \in Y\}\) with \(X \in \mathbb{R}^d\) are \(d\)-dimensional feature vectors and \(Y \in \mathbb{N}\) are discrete labels/classes, the classification problem is to learn/train a model \(H(x) : X \rightarrow Y\) that takes a sample \(x\) as input and outputs its probabilities of being in each class. The dataset is defined by a set of “sample-label” pairs \((x, y)\) with \(x\) a \(d\)-dimensional feature vector (or “input”) and \(y\) its associated label. Suppose a classification problem has \(C\) unique labels (or classes), the model \(H(x)\) will be learned to output a \(C\)-dimensional probabil-
ity vector indicating the probabilities of $x$ being in each of the $C$ possible classes, the class with the highest probability is often chosen as the predicted class of $x$.

In supervised learning, a set of data should be collected and labeled in advance for model learning. For training and validation, collected data is often divided into training set ($X_{\text{train}}, Y_{\text{train}}$) and validation set ($X_{\text{val}}, Y_{\text{val}}$), with the training set is used for model learning and the validation test for model evaluation. Once a classification model is learned on training data, it can then be used to make predictions on a test set of samples ($X_{\text{test}}$). The model’s prediction performance on test data indicates its generalization performance, which is often evaluated by the discrepancy between the predicted labels ($Y_{\text{pred}}$) and the ground-truth labels ($Y_{\text{test}}$). A model that overfits the training data usually exhibits poor generalization performance on the test data. In the next two subsections, we will introduce the two types of classification models exploited in this thesis: deep neural networks and random forests.

2.1.2 Deep neural networks

Deep neural networks (or DNNs) are artificial neural networks (or ANNs) of deep architectures with at least 2 layers. ANNs are models with computational units organized in a layer-wise architecture that is vaguely inspired by the structure of human brain. In ANNs, computational units or nodes are called artificial neurons which loosely model the neurons in a biological brain, and are connected layer by layer. Like the synapses in a biological brain, artificial neurons take signals from previous layer, process them, then pass on to the next layer. Figure 2.1 illustrates the architecture of a 2-layer ANN. The depth of a network only counts for layers with trainable parameters, thus input layer does not count as a layer. The concept of “deep” in DNNs refers to the ability of the network to go deep (to even thousands of layers), facilitated by modern designs of architectures and efficient training techniques such as stochastic gradient descent (SGD) [14] and backpropagation [108, 218].

Although DNN only starts to drawn significant attention from the past decade, it has a long history back to 1950s. The first and simplest network is the perceptron (or a single layer perceptron) invented by Frank Rosenblatt in 1958 [170] for
2. BACKGROUND

Figure 2.1: A 2-layer artificial neural network (or a feedforward neural network), which consists of an input layer with 3 neurons, a hidden layer with 4 neurons and an output layer with 2 neurons. It takes 3-dimensional feature vectors as inputs and outputs a 2-dimensional probability vector.

pattern recognition. It is a linear classifier with only a single layer that directly feeds the input vectors to the outputs via a linear transformation. Although the perceptron showed promising results on learning simple linear functions, it was quickly proved to have limitations on complex functions such as the XOR function [135]. This had caused the field of neural network research to stagnate for many years until it was recognized that a neural network with more than one layers is capable of learning complex nonlinear functions [32].

With the development of efficient training techniques such as backpropagation and gradient descent [108, 165, 175, 218], neural networks with more layers and complex inner connections are proposed nowadays to deal with learning problems of varying forms and difficulties. For example, convolutional neural networks (CNNs or ConvNets) for image/video related learning problems, recurrent neural networks (RNNs) for sequential learning problems such as time series analysis and speech recognition [196, 78], and autoencoders for encoding and denoising [62, 211]. These networks differ from each other in their internal structures, training strategies and the ability to deal with different forms of data. In the following two subsections, we will introduce the two types of neural networks investigated in this thesis: feedforward networks and convo-
2.1. PRELIMINARIES

2.1.2.1 Feedforward neural networks

Feedforward neural networks (also known as multilayer perceptrons (MLPs)) are one class of ANNs with only feedforward connections, which is different to RNNs where neurons have feedback connections to themselves. As illustrated in Figure 2.1, feedforward networks have three types of layers including input layer, hidden layer and output layer, and neurons in a layer are fully connected to all the neurons in the next layer. The input layer receives feature vectors as inputs, and feeds forward the inputs directly to hidden layers where linear transformations followed by nonlinear activations are applied successively layer by layer. The output layer receives transformed features from its previous layer, and outputs a probability vector. The probability vector consists of normalized values in range $[0, 1]$ indicating the probability of the input vector being in each class, and it is usually computed by a linear transformation with a normalization to probabilities.

The goal of a feedforward neural network is to approximate some function $H^*$ that defines the mapping from input domain $X$ to class domain $Y$:

$$H^* : X \rightarrow Y. \quad (2.1)$$

Training a feedforward neural network $H(x; \theta)$ is to learn the optimal parameters $\theta$ that best approximate the underlying function $H^*$. For simplicity, in this thesis, we omit the notion of parameters $\theta$ from $H(x; \theta)$ and its decompositions. Taking each layer as a transformation function, a $L$-layer feedforward neural network can be formulated as:

$$H(x) = h^L \circ h^{L-1} \circ \cdots \circ h^2 \circ h^1(x), \quad (2.2)$$

where $h^i \circ h^{i-1} = h^i(h^{i-1}(\cdot))$ is the composition of two functions. Therefore, feedforward neural network can be seen as a chain of transformations with information flow from input layer to output layer via a series of intermediate computations, and the length of the chain gives the depth of the network.
Here, we introduce three commonly used activation functions for hidden layer transformations. The transformation function $h(\cdot)$ represented by a hidden layer can be further decomposed into a linear transformation and a non-linear activation $\sigma$ as:

$$h(x) = \sigma(w^\top x + b),$$  \hspace{1cm} (2.3)

where $w$ and $b$ are the weight vector and the bias variable. The three commonly used activation functions are illustrated in Figure 2.2. The sigmoid nonlinearity is defined by:

$$\text{sigmoid}(x) = 1/(1 + e^{-x})$$  \hspace{1cm} (2.4)

Sigmoid function squashes a real-valued number into the range of $(0, 1)$ with large negative values be transformed to values close to 0 and large positive values be transformed to values close to 1. The sigmoid activation has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1) [110]. However, it has two major drawbacks: 1) it easily becomes saturated and leads to the vanishing gradient problem; and 2) its activations are not centered around zero which can cause unstable gradient-based parameter update [58, 59]. The tanh activation is defined by:
tanh(\(x\)) = 2\text{sigmoid}(2x) − 1 \hspace{1cm} (2.5)

The tanh function squashes a real-valued number into the range of \((-1, 1)\) which is zero-centered, but it still suffers the saturation and vanishing gradient problem like the sigmoid function \[58\]. The Rectified Linear Unit (ReLU) is defined by:

\[
\text{ReLU}(x) = \max(0, x) \hspace{1cm} (2.6)
\]

ReLU has become popular for training modern networks of very deep architectures \[59, 61\]. By simply clipping the negative values to 0, ReLU not only helps the convergence of stochastic gradient descent but also is computationally efficient, which makes training of very deep networks possible \[101\]. But ReLU does not squash the positive activations, thus may produce extremely large gradients. Large gradients may cause some neurons “overlearned” at intermediate steps and will never be activated again. This is known as the “dying ReLU” problem. Although several generalizations of ReLU have been proposed to address the dying ReLU problem such as Leaky ReLU \[227\], parametric ReLU (PReLU) \[227\], exponential linear units (ELU) \[28\] and Maxout \[64\], they are not as efficient as ReLU.

Training a neural network is a parameter update (or fitting) process to find the optimal parameters \(\theta\) of the network such that the model distribution \(\hat{p}(y|x; \theta)\) best approximates the data distribution \(p(y|x)\). Following the principle of maximum likelihood, the loss function (also known as the cost or objective function) is the cross-entropy between the empirical data distribution \(p(y|x)\) and the model distribution \(\hat{p}(y|x; \theta)\):

\[
J(\theta, x, y) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{y} y \log \hat{p}(y|x_i; \theta)
\]

\[
= -\frac{1}{N} \sum_{(x,y) \sim p} p(y|x) \log \hat{p}(y|x; \theta) \hspace{1cm} (2.7)
\]

\[
= -\mathbb{E}_{(x,y) \sim p} \log \hat{p}(y|x; \theta),
\]
which is simply the expectation of the negative log-likelihood of the model distribution, and \( N \) is the total number of data samples and \( C \) is the number of classes. Although classification DNNs are trained with the cross entropy loss in general, the exact form of the loss function varies from model to model, depending on the specific formulation of \( \hat{p}(y|x; \theta) \) by the network. One commonly used transformation at the output layer is the softmax function:

\[
h^L(x) = \text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \tag{2.8}
\]

where \( z \) is a \( C \)-dimensional logits vector (\( C \) is the total number of classes) obtained via a linear transformation on the last hidden layer:

\[
z = w^T h^{L-1}(x) + b. \tag{2.9}
\]

The softmax function can be regarded as a generalization of the sigmoid function (see Equation (2.4)).

The output transformation can also be linear, in which case, model output is just a linear transformation of the last hidden layer, e.g. \( H(x) = z \). Linear output layers are often used to produce the mean of a conditional Gaussian distribution:

\[
p(y|x) = \mathcal{N}(y, H(x), \delta^2). \tag{2.10}
\]

This is to assume that the data distribution \( p(y|x) \) is a Gaussian distribution with some fixed variance \( \delta \), and the network learns the mean of the Gaussian. In this case, the cross entropy loss is:

\[
J(\theta, x, y) = -\frac{1}{N} \sum_{(x,y) \sim p} p(y|x) \log \hat{p}(y|x; \theta) = \log \delta + \frac{1}{2} \log(2\pi) + \sum_{(x,y) \sim p} \frac{(y - H(x))^2}{2\delta^2}. \tag{2.11}
\]

In this case, the loss \( J(\theta, x, y) \) is equivalent to the mean square error by omitting the first two constant terms and scaled up by a factor of \( \delta^2 \) that does not depend
2.1. PRELIMINARIES

on θ:

\[
J(\theta, x, y) = \frac{1}{2} \sum_{(x,y) \sim p} (y - H(x))^2. \tag{2.12}
\]

While softmax-based cross entropy is the most commonly used loss, there are also other loss functions exist in the literature, such as hinge loss [24], mean absolute error [61, 131], center loss [217] and contrastive loss [123]. Following the maximum likelihood principle, different loss functions encourage different properties of the learned network. In practice, the loss function is often coupled with a regularization term such as \(L_2\) weight regularization to learn simpler hypothesis with better generalization [61].

A neural network can be trained by gradient decent with backpropagation. Network training is an optimization process to find the optimal \(\theta\) that gives the best approximation of the true underlying conditional distribution \(p(y|x; \theta)\). Training can be done iteratively, with each iteration includes a forward pass and a backpropagation. During forward pass, the input \(x\) or a minibatch of inputs are fed forward through the network to compute the output probability distribution, based on which the loss/error is calculated. During backpropagation, the error is back-propagated layer by layer from output layer to the first hidden layer, along with the weights/parameters \(\theta\) in each layer adjusted towards the direction of minimizing the error. A forward pass followed by a backpropagation is called an iteration, and one complete pass over all training samples through the network is call an epoch. Some commonly used optimizers for parameter update includes stochastic gradient descent (SGD) [14], RMSprop [206], Adagrad [42], Adadelta [232] and Adam [97, 162]. These optimizers differ from each other in the optimization rules used for parameter update, which leads to different stability and rates of convergence.

The time it takes to train a neural network depends on the complexity of the network (e.g. the number of trainable parameters), the scale of the training data and other hyperparameters such as learning rate. On large-scale datasets such as ImageNet [37], it may take days or even months to train a very deep neural network with thousands of layers [69, 176]. Powerful GPUs can be used to speed up numerical computing thus greatly reduces training time [101]. It is often difficult for feedforward networks to go very deep, as the number of
parameters increases exponentially to the depth of the network. In the next subsection, we will introduce convolutional networks that have better scalability to very deep architectures, and are particularly effective on learning image data.

2.1.2.2 Convolutional neural networks

Convolutional neural networks (CNNs or ConvNets) are one class of ANNs that apply convolutional operations between layers, in contrast to the linear operations in feedforward networks. As illustrated in Figure 2.3, CNN organizes neurons into 3D volumes at each layer, which is different to the flat layers in feedforward networks. The 3D architecture is initially designed to match image data that has 3 dimensions: height, width and channel (RGB colors), and to allow efficient convolutional operations [110].

In feedforward networks, neurons of a layer are fully connected to the next layer, and the inter-layer transformation is a linear operation followed by an activation nonlinearity. In CNNs, neurons of a layer are sparsely connected to the next layer via convolution kernels (or filters), and the inter-layer transformation is a convolution operation followed by an activation nonlinearity. A convolutional network still express a single differentiable score function mapping input to a probability vector, and the loss functions for feedforward networks introduced in Section 2.1.2.1 still apply for CNNs. CNNs can also be trained using gradient descent with backpropagation, like feedforward networks. In the following paragraphs, we will introduce the intuition behind CNNs and their unique properties that are different to feedforward networks.

In [108, 110], LeCun et al. developed a 5-layer CNN known as the LeNet-5 for recognizing handwritten digits, and showed that convolutions as a part of a neural network can lead to several desirable properties such as shift invariance, translation invariance and training efficiency. Due to these advantageous properties, CNNs have drawn significant attention and have become deeper and wider in the last two decades. In the history of CNN, many classic architectures have been developed, such as VGGNet [186], AlexNet [176], ResNet [70] and InceptionNet [200], and some of them have achieved surpassing human-level
2.1. PRELIMINARIES

Figure 2.3: **Left:** A 3-layer feedforward neural network where neurons are flattened within each layer. **Right:** A convolution network arranges neurons into 3D volumes (width, height and depth), as visualized in the hidden layer.

Figure 2.4: **Left:** Kernel based convolution on a 3D input volume produces a 3D output volume with depth equals to the number of kernels. **Right:** A 2D example of kernel operation. The kernel scans each element (black) on the input plane (blue) to apply a dot product between the kernel and the local patch (orange+black) centered at the reference element (black). For each element of the input, the kernel operation produces one single value. Zero padding (zero elements outside the input plane) is often used to ensure the overall output volume has the same shape as the input volume.

performance on image classification tasks [70]. The latest versions of ResNet are more than a thousand layers deep.

The three main types of layers in a typical CNN are: convolutional layer,
pooling layer, and fully-connected layer. Convolutional layer is the core building block of CNN that applies kernel operations on 3D input volume to produce a 3D output volume, as illustrate in Figure 2.4. The kernel is inspired by the receptive field of human visual system to capture the local spatial information contained in an image. Convolutional kernels have the same number of dimensions as the input volume, and the commonly used 3D kernels are of shape $height \times width = 3 \times 3$ or $5 \times 5$, and the same depth as the input volume. As shown in the 2D example in Figure 2.4, a kernel scans each input element along the height and width directions to apply dot product between the kernel weights and the local patch centered at the element.

With zero padding, a typical technique to avoid dimension reduction, each kernel produces a 2D output slice of the same height and width as the input, with the depth dimension reduced during the convolution. The number of kernels determines the depth of the output volume, e.g. the number of 2D slices produced by the kernels. The kernel shares weights for all its associated convolutions, which is known as the “parameter sharing” scheme of CNN. Compared to feedforward networks, CNN parameter sharing greatly reduces the number of learnable parameters, thus makes training deep networks more efficient. After convolutional transformation, the values will be further transformed by a nonlinearity such as the widely used ReLU function.

In CNNs, pooling layers are often used periodically in-between successive convolutional layers. Pooling is a downsampling operation with filters to downsample local paths to leave only the maximum value within each patch. A typical max pooling filter is of size $2 \times 2$ (only on the height and width dimensions), and works like convolution filters scanning across the height and width directions. A trike of 2 is often used to skip one element between two successive operations to mitigate information loss caused by the downsampling. Max pooling greatly reduces model complexity by shrinking the output volume while keeping the most significant features. Pooling layers have been found to improve generalization and training efficiency, thus have become an essential part of CNN architecture [61, 101]. Fully-connected layers in CNNs are essentially the same as flattened layers in feedforward networks.

There also exist other popular operations that have been used in CNNs to
2.1. PRELIMINARIES

Figure 2.5: An example of a 7-layer CNN for image recognition. The internal representations are illustrated across layers of the network. The output layer (FC layer) produces a probability distribution of the input image being in each class [114].

improve generalization, robustness or training efficiency, such as batch normalization (BatchNorm) [87] and dropout [188]. BatchNorm normalizes the values in a hidden layer to a more regularized distribution of zero mean and unit variance, and is often inserted before the ReLU activation. BatchNorm improves the robustness of CNNs to distribution shift between training and test data, and helps the convergence of the network. Dropout is a regularization technique that randomly sets a certain proportion (50% is a common setting) of values in a layer to zero. Dropout brings randomness into the learning process, so as to avoid overfitting and improve generalization. Dropout is a training technique, and is often turned off at test time to avoid information loss.

Same to feedforward networks, a CNN also defines a differentiable loss function, and can be efficiently trained using gradient descent. The selection of loss function is more application-based and is independent of network architecture. Therefore, for CNNs on supervised learning problems, the typical loss function is still the cross entropy loss introduced in Section 2.1.2.1. An example of a 7-layer CNN can be found in Figure 2.5: CONV-CONV-POOL-CONV-CONV-POOL-CONV-CONV-POOL-FC. As visualized, different layers
2. BACKGROUND

Figure 2.6: An illustration of the tree construction process of a random forest with four decision trees. To build each tree, the algorithm randomly selects a subset of samples and features from training data, and randomly selects a feature to construct each node. Within each node, the optimal partition of the samples is taken as the split condition of the node.

of the CNN learn different levels of abstractions of the underlying concept ‘car’, where shallow layers learn simpler patterns such as lines and corners while deeper layers learn complex patterns and shapes. We can also see the clipping effect of ReLU nonlinearity and the down-sampling effect of pooling layers.

2.1.3 Random forests

In this subsection, we introduce one of the popular traditional (relative to modern DNNs) machine learning models: random forests (RFs, which are also known for random decision forests). The RF model is an ensemble of decision tree classifiers developed in the 1990s [17, 76, 77]. The basic principle behind RFs is that an ensemble of weak classifiers together works better than any of the weak classifiers individually. The weak classifier of RF is the decision tree classifier [120, 160, 161, 177]. In the next few paragraphs, we introduce the construction and interpretation of RFs on the basis of decision tree model.

A RF model can be built in a iterative manner until a certain number of
2.1. PRELIMINARIES

Figure 2.7: An example of a random forest with two decision trees of maximum depth 2. The first tree is built based on $x_1$ and $x_2$ while the second tree is built based on $x_2$ and $x_3$. The leaf nodes indicate the class, and the intermediate nodes from root to leaf determine which branch an input falls into based on the split condition of the node.

decision trees are constructed, with a decision tree being built within each iteration. The overall process is illustrated in Figure 2.6. The total number of decision trees and the maximum depth of individual trees are hyperparameters that need to be set in advance. To construct a decision tree, the algorithm selects a random subset of features as well as a random subset of samples from the training data, then it iteratively chooses a feature from the feature subset as the root of the tree to partition the sample subset, and moves downwards to construct its child nodes until there is no sample to partition or the maximum depth of the tree is reached. The feature and its value at which the samples are partitioned is set as the split condition of the node. Gini impurity and information gain \cite{76, 77} are two commonly used partition criteria to find the optimal split value of the feature that best differentiates the class distributions after the partition. After a decision tree is constructed, the process is repeated until the total number of decision trees are constructed. An example of RF with two decision trees and three features ($x_1$, $x_2$, and $x_3$) can be found in Figure 2.7: the algorithm randomly selects two features at a time to build a decision tree of maximum depth 2 until it builds all the two trees.

For prediction, a test input will be examined by each decision tree of a RF model. Each tree will make a prediction on the input based on the class of the
leaf node it falls into, which can be regarded as one vote for the predicted class. And the class wins the majority votes from all decision trees in the RF will be taken as the final predicted class of the test input.

Built upon decision trees, RFs have better interpretability than other traditional classification models such as logistic regression and support vector machines (SVMs) [72]. This is because the decision rule defined by the path from root to a leaf node can be directly interpreted as a set of properties of the corresponding class. Meanwhile, RFs are very fast to train. Due to these reasons, RFs are still widely used for some classification problems nowadays, although DNNs start to dominate some learning tasks such as speech recognition [74], image classification [101] and face recognition [155].

2.2 Adversarial perturbations and deep neural networks

In this section, we introduce the concept of adversarial perturbations that can easily fool DNNs to make misclassifications with small perturbations, despite the fact that DNNs are often capable of learning very complex patterns. We also review existing works in the literature on 1) understanding adversarial perturbations and DNNs, 2) attacking methods that craft adversarial examples to fool DNNs, and 3) defense methods to secure DNNs against adversarial attacks. In this thesis, we investigate the adversarial vulnerability of DNNs, and provided a novel explanation of this phenomenon based on a characterization of the dimensional properties of adversarial examples and their local subspaces.

2.2.1 Adversarial perturbations

With the significant improvements in DNN computational power, modern DNNs are becoming deeper and deeper with more than a thousand of layers and millions of parameters. In many applications such as image classification [101], face verification [155] and speech recognition [74], DNNs have demonstrated superior performance than traditional models such as RFs and SVMs. On some
2.2. ADVERSARIAL PERTURBATIONS AND DEEP NEURAL NETWORKS

Figure 2.8: An example of adversarial perturbations and adversarial example. Adding small adversarial perturbations (middle) to a clean image (left) can fool a CNN to misclassify the perturbed image (which is known for “adversarial example”) to ‘gibbon’ with 99% confidence, though the two images are very similar to human eyes [63].

tasks such as image classification [70] and playing the game of Go [185], DNNs even achieved surpassing human-level performance.

Despite their current success, recent studies have found that DNNs are vulnerable to adversarial perturbations, even though those perturbations are small, imperceptible to human eyes in the case of images [22, 63, 145, 202, 205]. Adversarial perturbations are test-time input perturbations generated by an attacking method (or an adversary) that can be used to perturb normal test samples to mislead a DNN (pretrained on normal samples) to make misclassifications. This phenomenon is called the “adversarial phenomenon” and the adversarially perturbed samples are called “adversarial examples” [63]. Figure 2.8 and 2.9 demonstrate several examples of adversarial perturbations and adversarial examples. In Figure 2.8, a pretrained CNN misclassified the adversarially perturbed ‘panda’ image (right image) to ‘gibbon’ with 99% confidence, though it looks no different to the original ‘panda’ image (left image) [63] as the amount of perturbations (middle image) is very small. More adversarial examples are illustrated in Figure 2.9, where handwritten digits were adversarially perturbed to fool a CNN classifier with high success rate [63, 202].

Although adversarial perturbations can be regarded as one type of input noise, they differ from random input noise in various aspects. First, they are test-time noise, meaning that they are not normal input noise in the training
2. BACKGROUND

Figure 2.9: Adversarial handwritten digits (bottom row) perturbed from normal digits (top row). Although a CNN can correctly recognize the normal digits as their true classes (second row), it misclassify all the adversarial digits to incorrect classes (third row).

data and thus not involved in the network training. Second, the perturbations are intentionally-designed to cause misclassifications rather than random. Third, adversarial perturbations are particularly small, imperceptible to human eyes in the context of images.

The adversarial vulnerability of DNNs has attracted enormous attention since its first discovery by Szegedy et al. [202]. Studies have shown that adversarial examples can be easily generated using adversarial techniques [11, 22, 63, 103, 145, 202]. Moreover, adversarial examples generated from one DNN can be used to attack other DNNs, even though these DNNs are of different architectures and trained separately [124, 137, 151, 207]. Meanwhile, recent works have demonstrated that physical-world adversarial examples are also possible, and these examples can fool real-world AI systems to make incorrect or dangerous decisions [45, 103, 152]. For example, in [45], it showed that adversarial graffiti on a ‘Stop’ sign can fool a DNN to recognize it as a ‘Speed Limit 45’ sign, and this could mislead a self-driving car to speed up at a stop sign. Therefore, the DNN vulnerability to adversarial perturbations has become one of the major security concerns for applying DNNs in real-world applications.

Szegedy et al. argued that the existence of adversarial perturbation is due to the high nonlinearity of deep networks, as most of kernel methods are nonlinear [202]. The DNN nonlinearity tends to result in discontinuous mapping from input domain $X$ to output domain $Y$, thus causes the existence of densely
scattered subspaces that stay close to normal data submanifold, yet have unpredictable class distributions \[203\]. It is further argued in \[63\] that the adversarial phenomenon is actually caused by the linear nature of deep networks. This is based on the fact that several commonly used transformations such ReLUs \[127\] and maxout \[64\] are intentionally designed to behave in very linear ways. They also demonstrated that for a linear classifier, as the input dimension increases, small changes in individual dimensions can sum up to significant change in the output. However, Tanay and Griffin \[205\] showed that the linear explanation also has limitations, and proposed a “boundary tilting” perspective of explanation: adversarial examples only exist when the classification boundary lies close to the submanifold of sampled data. Their reasoning was based on the observation that some linear models do not have adversarial examples. While these linear and nonlinear explanations appear to contradict each other, a comprehensive understanding of DNNs and their vulnerability to adversarial examples is still need to be established \[6, 21\].

In this thesis, we tackle this challenge by characterizing the dimensional properties of adversarial subspaces in the vicinity of adversarial examples, and provide a new perspective of understanding to the community. Our dimensionality based analysis of adversarial subspaces not only motivates new directions of effective adversarial defense, but also opens up more challenges for developing new attacks to better understand the vulnerabilities of DNNs. In the next two subsections, we review existing attacking methods that have been proposed to craft adversarial examples and current defense strategies developed to secure deep networks against such attacks.

### 2.2.2 Adversarial attack

A wide range of approaches have been proposed for the crafting of adversarial examples to compromise the performance of DNNs; first, we summarize a selection of such works. The Fast Gradient Method (FGM) \[63\] directly perturbs normal input by a small amount along the gradient direction. The Basic Iterative Method (BIM) is an iterative version of FGM \[103\]. One variant of BIM stops immediately once misclassification has been achieved with re-
spect to the training set (BIM-a), and another iterates a fixed number of steps (BIM-b). For image sets, the Jacobian-based Saliency Map Attack (JSMA) iteratively selects the two most effective pixels to perturb based on the adversarial saliency map, repeating the process until misclassification is achieved [153]. The Optimization-based attack (Opt), arguably the most effective to date, addresses the problem via an optimization framework [22, 124]. We will review these works in detail below, before which we introduce the adversarial generation problem these adversaries attempt to solve.

We describe the adversarial generation problem in the context of image classification. Let input domain be $X \in \mathbb{R}^d$, class domain be $Y \in \{0, 1\}^C$, a deep network $H(x) : X \rightarrow Y$ is a functional mapping of the $d$-dimensional input domain $x$ to a $C$-dimensional discrete class domain. Denote the loss function of a network by $J(\theta, x, y)$ (such as the cross entropy loss in Equation (2.7)), with $\theta$ the parameters of the network, $x$ the input image and $y$ the class label associated to $x$. Given a test image $x$ with class $y$, the goal of an attacking method (also called an “adversary”) is to craft a new image $x_{\text{adv}}$ such that $H(x_{\text{adv}}) \neq y$ and the amount of perturbation is minimized:

$$\min \|x_{\text{adv}} - x\|_p \text{ s.t. } H(x_{\text{adv}}) \neq y,$$  \hspace{1cm} (2.13)

where $\|\cdot\|_p$ is the $L_p$ norm defining the amount of perturbation. Some commonly used $L_p$ norms are $L_0$, $L_2$ or $L_\infty$.

Equation (2.13) defines the so called “untargeted attack” problem, meaning that the adversary only needs to perturb the input $x$ to an arbitrary class that is incorrect. The attack can also be “targeted”, in which case input $x$ will be perturbed into a particular incorrect class $y_{\text{target}} \neq y$. Accordingly, the targeted adversarial generation problem is defined as:

$$\min \|x_{\text{adv}} - x\|_p \text{ s.t. } H(x_{\text{adv}}) = y_{\text{target}} \neq y.$$ \hspace{1cm} (2.14)

In general, targeted adversarial examples are more difficult to generate than untargeted adversarial examples.

Various attacking methods have been proposed to craft adversarial exam-
2.2. ADVERSARIAL PERTURBATIONS AND DEEP NEURAL NETWORKS

Adversarial examples to attack DNNs. They differ from each other in the way to solve the above defined adversarial generation problems in Equations (2.13) and (2.14). It should be noted that adversarial generation is different from network training. Adversarial generation is a post-processing process on a pretrained network. In adversarial generation, the input \( x \) will be updated instead of model parameters, which is in contrast to network training where the parameters \( \theta \) are updated. Moreover, adversarial generation aims to maximize the loss so as to fool the network to make mistakes, while network training aims to minimize the loss.

Fast Gradient Method (FGM): FGM is an untargeted attacking method which perturbs each feature of an input \( x \) by a small magnitude towards maximizing the prediction loss \( J(\theta, x, y) \) [63]. Formally, it generates adversarial examples as follows:

\[
x_{\text{adv}} = x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))
\] (2.15)

FGM is a fast method which only perturbs the input once, with a fixed amount of perturbations controlled by \( \epsilon \). Therefore, it is not guaranteed to successfully perturb the input to an adversarial class (i.e. \( H(x_{\text{adv}}) \neq y \)). The success rate can be improved by increasing the perturbation magnitude \( \epsilon \), although this may result in large perturbations that are no longer imperceptible to human observers. FGM is widely used in the community as a benchmark attacking method or a baseline evaluation method for new defense strategies, due to its high efficiency (only need one step gradient-based update) and high attacking success rates on MNIST [109] and CIFAR-10 [100] datasets.

Basic Iterative Method (BIM): BIM is an iterative version of FGM that iteratively applies FGM for some fixed steps with smaller \( \epsilon \) [103]. Formally, it is:

\[
x_{\text{adv}}^0 = x \\
x_{\text{adv}}^i = \text{clip}_{x, \epsilon}(x_{\text{adv}}^{i-1} + \epsilon \text{sign}(\nabla_{x_{\text{adv}}^{i-1}} J(\theta, x_{\text{adv}}^{i-1}, y))).
\] (2.16)

Here, \( \text{clip}_{x, \epsilon}(\cdot) \) is an element-wise clipping to ensure the perturbed sample \( x_{\text{adv}} \) remains a valid input within a \( \epsilon \)-ball surrounding \( x \). BIM provides more flexible control over the perturbation process while remains efficient as it can stop
whenever misclassification $H(x_{adv}) \neq y$ is achieved (BIM-a version), or after a fixed number of iterations (BIM-b version). BIM was demonstrated to be more effective than FGM on natural images such as CIFAR-10, ImageNet [37, 103].

**Jacobian-based Saliency Map Attack (JSMA):** JSMA is a targeted attack with a $L_0$ form of objective [153]. Given a targeted adversarial class $y_{target} \neq y$, it iteratively selects the two most effective pixels to perturb a small magnitude of $\epsilon$ until the target class is achieved. The perturbed pixels are selected based on the *saliency map* defined for each pixel:

$$S(x, y)[i] = \begin{cases} 
0, & \text{if } \frac{\partial H_{y_{target}}(x)}{\partial x_i} < 0 \text{ or } \sum_{y' \neq y_{target}} \frac{\partial H_{y'}(x)}{\partial x_i} > 0 \\
\left| \frac{\partial H_{y_{target}}(x)}{\partial x_i} \right| \left( \sum_{y' \neq y_{target}} \frac{\partial H_{y'}(x)}{\partial x_i} \right), & \text{otherwise}
\end{cases} \quad (2.17)$$

Within each iteration, the pixels pair $(i, j)$ with the maximum $S(x, y)[i] + S(x, y)[j]$ is selected to perturb a constant amount of $\epsilon$. This process is repeated until $H(x_{adv}) = y_{target}$. A pixel-wise clipping is also applied to $x_{adv}$ at the end of each iteration. When tested on MNIST, JSMA demonstrated high success rates with only a few number of perturbed pixels.

**Optimization Based Attack (Opt.):** Opt. attack refers to the class of attacking methods that generate adversarial examples using some optimization based framework. One representative of this class of attacks is the attack proposed in [22], which is also known as the Carlini-Wagner (CW) attack. As the optimization problems defined in Equations (2.13) and (2.14) are hard to solve, Opt. attack transforms the problems to more solvable forms so that can be solved by an alternative optimization framework using gradient descent [22, 124].

Instead of directly optimizing on input $x$, Opt. attack applies a change-of-variables to introduce a new variable $w$ and optimize over the new variable:

$$\delta = \frac{1}{2}(\tanh(w) + 1) - x, \quad (2.18)$$

where $\delta$ is the magnitude of perturbation. The transformed optimization problem on $w$ is:

$$\text{minimize } \|\delta\|_2^2 + \alpha \cdot \ell(x + \delta) \quad \text{s.t. } x + \delta \in [0, 1]^d. \quad (2.19)$$
Here $\ell(\cdot)$ formulates the misclassification constraint $H(x_{adv}) \neq y$ or $H(x_{adv}) = y_{target}$ (see Equations (2.13) and (2.14)), and $\alpha$ is a constant balancing between the amount of perturbation and the adversarial strength. Several options have been proposed for $\ell(\cdot)$, among which the commonly used one for targeted attack is:

$$
\ell(x_{adv}) = \max(\max\{z_i(x) : i \neq y_{target}\} - z_{y_{target}}(x), -\kappa)
$$

(2.20)

where, $z_i(x)$ is the logits (outputs before the softmax transformation) of the network for class $i$, $y_{target}$ is the adversarial class and $\kappa$ is a parameter controls the confidence of misclassification.

Depends on the specific form of $\ell(\cdot)$, Opt. can be either targeted or untargeted, and can work with different $L_p$ norms including $L_0$, $L_2$ and $L_{\infty}$ [22]. The Opt. attack is one of the state-of-the-art attacks that can craft adversarial examples with higher success rates and smaller distortions [6, 21]. Meanwhile, there are also other attacks such as Deepfool [138], Projected Gradient Descent (PGD) [129] or variants of above introduced attacking methods such as JSMA-F [22].

### 2.2.3 Adversarial defense

A number of defense techniques have been proposed to secure DNNs against adversarial attacks, including adversarial training [63, 187], defensive distillation [154], gradient masking [66, 141], and feature squeezing [228]. However, these defenses can generally be evaded by Opt attacks, either wholly or partially [21, 71, 112, 113].

One straight forward defense strategy is the adversarial training, that is, mixing adversarial examples into training data to allow deep networks learn to differentiate these examples. Considering the superior learning capability of deep networks, this can be expected to be an effective approach. It was suggested in [63] that taking adversarial examples as augmented data has the potential to train robust networks. Following this direction, methods have been investigated to learn robust networks with augmented adversarial examples generated at an intermediate step during training [85, 129].

Recall standard network training is to find the optimal parameters $\theta^*$ such
Adversarial training is to solve a min-max optimization problem:

$$
\theta^* = \arg \min_{\theta} \mathbb{E}_{x \in X} \left[ \max_{\delta \in [-\epsilon, \epsilon]^d} J(\theta, x + \delta, y) \right],
$$

(2.22)

where the internal maximization term defines a $d$-dimensional $\epsilon$-ball that allows adversarial perturbations to explore.

Adversarial examples used for training can be generated at each training iteration [136, 182] or at a separate re-training/finetuning step [65]. However, adversarial training depends on the specific attacking method used to generate adversarial examples (within the constrained $\epsilon$-ball). This greatly limits the diversity of the adversarial examples, thus adversarial training often provides marginal improvements over standard training [22].

Model smoothing techniques have also been investigated to reduce the non-linearity of deep networks, which thus can be expected to increase adversarial robustness [202, 205]. One such work is defensive distillation which trains a secured network by distilling information from a pretrained standard network [154]. Defensive distillation tends to produce gradients $10^{30}$ times smaller than standard training, which “kills” the gradients that can be exploited by an adversary. Unfortunately, defensive distillation has also been compromised by the Opt. attack [22].

Regularization techniques can also be used to mask or kill the gradients. For example, regularizing the Frobenius norm of the Jacobian matrix [66], or regularizing the activation map [141]. However, adversarial examples can still be generated on a substitute vulnerable network, then transferred to attack regularized models [124, 22]. Moreover, strong regularizations may significantly reduce network capacity thus sacrifice prediction performance.

### 2.2.4 Adversarial detection

Considering the great difficulty in adversarial defense, recent works have instead focused on detecting adversarial examples as the first step towards se-
cured deep learning. These works attempt to discriminate adversarial examples (positive class) from normal examples (negative class), based on features extracted from different layers of a DNN or artifacts of the learned representations. Works in this direction include detection subnetworks based on activations [134], cascade detectors based on the principal component analysis (PCA) projection of activations [115], augmented neural network detectors based on statistical measures, a learning framework that covers unexplored space in vulnerable models [171] [172], and logistic regression detectors based on KD and Bayesian Uncertainty (BU) artifacts [65]. However, a recent study by [21] has shown that these detection methods can still be compromised by strong adversarial attacks. We will review these works in detail below.

Detection subnetworks were introduced in [134] to detect adversarial examples based on the activation map of intermediate layers of a DNN. The detection subnetworks were attached to some layers of the network, and were trained directly on the normal and adversarial activation maps at the attached layers. Although this approach demonstrated high detection accuracy on the attacks used to train the detection subnetworks, it heavily relies on the type of adversarial examples available for training.

Denoising or dimension reduction techniques have also been suggested for detecting adversarial examples. Feature squeezing was introduced in [228] to differentiate adversarial examples from normal examples. Feature squeezing transforms inputs to a reduced space with lower dimensions. It has been found that the prediction discrepancy before and after squeezing can be used to detect adversarial examples. PCA [225] projections of inputs have also been explored to discriminate adversarial examples [10]. Particularly, the authors suggested that the larger PCA components of adversarial examples tend to possess higher weights than normal examples [73], and such difference can be used to discriminate adversarial examples. However, recent work argued that the PCA difference was caused by the smooth background of MNIST images (a large proportion of background is black), and it was not observed on complex datasets such as CIFAR-10 [21]. A cascade detector on the PCA projections of inner activations was investigated in [115]. However, it was also found not effective in discriminating adversarial examples on CIFAR-10 dataset [21].
As adversarial examples are not drawn from natural distributions, they may exhibit statistical differences to normal samples. This conjecture was confirmed in [65] where statistical artifacts such as maximum mean discrepancy and energy distance were used to detect adversarial examples. Kernel density (KD) is another statistical measure that has recently been proposed to discriminate adversarial examples [47]. The assumption behind KD is that normal samples from the same class lie densely on the data manifold (high density) while adversarial samples lie in more sparse regions off the data submanifold (low density). Given a point \( x \) with class \( y \) and a set of training samples from the same class \( X_y \), the Gaussian kernel density of \( x \) can be estimated by:

\[
\text{KD}(x) = \frac{1}{|X_y|} \sum_{s \in X_y} \exp \left( \frac{|h^{L-1}(x) - h^{L-1}(s)|^2}{\sigma^2} \right),
\]

where \( \sigma \) is the bandwidth parameter controlling the smoothness of the Gaussian estimation, \( h^{L-1}(x) \) is the representations at the last hidden layer. A simple logistic regression detector was trained over KD artifact to differentiate adversarial examples.

The Bayesian uncertainty (BU) was also suggested in [47] to detect adversarial examples. BU measures the prediction uncertainty of a DNN over an input sample. The intuition is that, in a randomized network, normal samples will receive similar results over multiple predictions regardless of the different randomization chosen for each prediction, however, adversarial examples tend to exhibit significant difference across different predictions. This method performs \( T \) times of predictions with the dropout operation turned on in a dropout randomized network, and utilizes the randomized probability distributions to estimate BU:

\[
\text{BU}(x) = \left( \frac{1}{T} \sum_{i=1}^{T} \| h^{L-1}(x) \|_2^2 \right) - \left( \frac{1}{T} \sum_{i=1}^{T} h^{L-1}(x) \|_2^2 \right),
\]

Combined with KD, this measure was shown a 85% - 93% ROC-AUC detection performance. Again in [21], it showed that the KD and BU detectors can also be evaded by a strong adversary.
In this section, we have reviewed current works in adversarial attack and defense/detection. To summarize, these works have explored the adversarial vulnerability of DNNs from the following three aspects: 1) understanding reasons behind the adversarial phenomenon, 2) generating adversarial examples to attack DNNs, and 3) developing defense or detection strategies to secure DNNs against adversarial attacks. Despite the significant number of existing works, new explorations are still in need to better understand DNNs and their vulnerability to adversarial perturbations. Different from existing works, in this thesis, we propose to characterize the dimensional characteristics of adversarial subspaces in the vicinity of adversarial examples. Particularly, in Chapter 3, we apply Local Intrinsic Dimensionality measure to perform such a dimensional analysis, and further demonstrate the possibility to utilize LID characteristics for the discrimination of adversarial examples. Our work not only provides new perspective of understanding of adversarial perturbations, but also motivates new defense strategies against adversarial attacks.

2.3 Feedback generation and simulation-based learning

In this section, we review existing works on feedback generation and its applications in simulation-based learning. We start this section with an introduction of feedback generation in supervised learning (SBL), then review current literature of SBL. Finally, we discuss existing approaches for the generation of feedback in SBL environments. In this thesis, we explore the novel use of adversarial techniques discussed in the above Section 2.2.2 to improve the generation of real-time feedback for a specific application of SBL: a virtual reality based surgical platform.

2.3.1 Feedback generation

In supervised learning, machine learning models are often trained to make predictions about the classes of input feature vectors such as the category of object contained in an image. However, for many complex learning tasks, mak-
ing class predictions is only a very first step towards some long-term learning goals. For example, in a virtual reality based driver training system, predicting a trainee’s driving skills to be ‘expert’ or ‘novice’ is a preliminary step towards the acquisition of expert-level driving skills [111, 168, 179]. To achieve such long-term learning goals, it often requires constant and interactive feedback from the learning system to support learning. In a broad sense, feedback is actionable instructions a learner receives from the system that can be followed to achieve some desirable outcomes. Feedback can play the role of a human expert to guide the trainee through a learning task with professional instructions.

The feedback generation problem is also known as the optimal action extraction problem [31]. It is a classical problem in artificial intelligence systems. Intelligent tutoring systems are one such class of systems that aims to provide immediate instruction or feedback to learners [12]. The other example is autonomous driving systems which take the surrounding environment as input and output feedback to the car to adjust the steering wheel [25]. In reinforcement learning systems such as mobile robot navigation, the hardware or software agent learns its behaviour based on reward feedback from the environment [197].

From the supervised learning perspective, feedback generation is to extract optimal action in terms of feature changes that can be applied to change the output from an undesired class to a desired class. For example, in customer relationship management, Yang et al. [229, 230] investigated several techniques to generate feedback (or actions) to change customers from an undesired status (such as ‘attritors’) to a desired one (such as ‘loyal’) with minimum cost.

Given a binary classification problem with one “positive” class as the desired outcome and one “negative” class as the undesired outcome, the feedback generation problem is to find the optimal action that can be taken to change a negative instance to a positive instance. Following the notations in Section 2.1.1, we denote a dataset by $X \in \mathbb{R}^d$ with $d$-dimensional features, an instance by $x$, the class label associated with $x$ by $y \in \{0, 1\}$ with 1 denoting the positive class and 0 denoting the negative class, and a prediction model learned over $X$ by $H(x)$. The feedback generation problem can then be defined as follows.

**Problem 2.3.1.** Given a prediction model $H(x)$ and a negative instance $x$, the problem
is to find the optimal action $A : x \rightarrow x'$ that changes $x$ to an instance $x'$ under limited cost $C$ such that $x'$ has the highest probability of being in the positive class:

$$\arg \max_{A : x \rightarrow x'} H(x')$$

subject to $\ell(x, x') \leq C$,

where, feedback $A : x \rightarrow x'$ involves one or multiple feature changes (value increase or decrease). For example, $A : x(x_1 = 0.2, x_2 = 0.3) \rightarrow x'(x_1 = 0.5, x_2 = 0.3)$ is “increase $x_1$ to 0.5”. $\ell(x, x')$ is the cost function measuring the potential cost of feedback $A$. The cost functions may vary in different applications [31, 229, 230]. For instance, in SBL systems, a commonly used cost is $\ell(x, x') = \|x - x'\|_0$, e.g. the number of changed features. This is because learners are easily distracted by complex instructions, simpler feedback with fewer instructions tend to increase its usefulness in practice. Therefore, the cost limit $C$ for SBL systems is often a small number like 1 or 2.

For certain models with continuous closed-form functions such as neural networks, the optimal direction of feature change can be found by a gradient-based method [63, 145, 202]. The feedback generation problem defined above shares certain similarities to the targeted adversarial generation problem defined in Equation (2.14). First, both of them aim to change a sample from one class to another. Second, they both target input perturbations to achieve class transitions. However, the two problems are not identical. Adversarial perturbations are small malicious input distortions that can be used to fool a classifier, while feedback is beneficial perturbations that can change the fundamental characteristics of a source sample to a target class, and the amount of change can be large. In other words, adversarial perturbations do not change the underlying true class of the original sample, but it can fool a network to believe there is a class change. Feedback, however, does change the true class of the original sample. Therefore, existing adversarial techniques are not directly transferable to feedback generation. Meanwhile, current DNN-based adversarial techniques cannot be applied to models without a closed-form function, such as RFs and SVMs, as the gradients cannot be computed directly. So far, no existing works in the literature have explored adversarial techniques for feedback generation.
In [31], it was proved that generating feedback from RF models is NP-hard, a problem cannot be solved within polynomial time. An integer linear programming (ILP) approach was proposed in [31] to solve the RF-based feedback generation problem by transforming it to a more solvable ILP problem. This approach converges to the global optimal solution if the problem itself is solvable. However, it is computationally expensive, thus not suitable for feedback generation in SBL environments where timely real-time feedback is required for interactive learning. In the following subsections, we will introduce SBL and feedback generation in such applications.

2.3.2 Simulation-based learning

Simulation-based learning (or SBL) is a modern learning methodology that involves practise and learning in a virtual (as opposed to physical) simulated environment [8, 105]. SBL platforms have been able to replicate real-world scenarios in an immersive and interactive fashion [192], and as such, provide realistic and versatile environments where individuals can learn new information, directly apply this information to simulated tasks, and master complex skills [12, 133]. With the development of virtual reality techniques [44, 147, 219], SBL platform has become an effective learning tool in a range of applications including surgery [68, 126, 169, 181, 221], military training [9, 30, 148, 149], driver/pilot training [36, 86, 158, 204, 212, 216], rehabilitation and therapy [146, 180].

Typically, SBL systems comprise of two essential components: 1) a simulation environment that provides an immersive and interactive learning experience, and 2) a feedback intervention system that supports knowledge/skill acquisition and decision making [222]. The simulation environment is created using technologies such as virtual [44] or augmented reality [7, 43]. Compared to traditional way of learning, SBL has many benefits: it is low-cost, low-risk, easily-accessible, repeatable, and versatile. In SBL, trainees can practice on multiple tasks designed with varying difficulty, while receive feedback on performance and instructions for better skill acquisition.

As an example of SBL platforms, a virtual reality based surgical simulator is shown in Figure 2.10 [126, 221]. It is a virtual reality simulator designed to
2.3. FEEDBACK GENERATION AND SIMULATION-BASED LEARNING

Figure 2.10: Left: A virtual reality based surgery simulator. Right: Anatomical structures of the temporal bone as rendered by the surgery simulator on the left.

train ENT (ear, nose, and throat) surgeons \[220\]. As shown in the left subfigure (in Figure 2.10), the simulator consists of a 3D temporal bone (bone beneath human ears, part of human skull) model running on a computer, and a haptic device (represented as a surgical drill in the virtual space), which provides tactile resistance to simulate drilling. Equipped with 3D glasses, the trainee can view the operating space in 3D, and drill the bone from the center area down to anatomical structures such as facial nerve and dura as illustrated in the right subfigure of Figure 2.10. The surgery simulator allows the practice of ear-related surgery such as cochlear implant surgery, where surgeons perform procedures to remove disease and improve hearing \[235\]. This often involves drilling the temporal bone or operating on the middle or inner ears.

Driving simulators have been designed for the training of new drivers \[204, 212, 216\]. Researches have proved that driving simulators are practical and effective tools for the learning of driving skills, and skills learned in simulated environments are transferable to real road conditions \[75\]. SBL also allows the practising of highly specialized tasks such as flight an aircraft or a military helicopter \[158\]. Task oriented simulations have also been used to provide effective treatment for people with psychological disorders or specific phobias such as fear of heights or public speaking. \[146\].

Supporting the learning process through interactive feedback is important \[12\]. Appropriate and timely feedback intervention increases learning motiva-
tion, facilitates skill acquisition/retention, and reduces the uncertainty of how a student is performing [34]. Although SBL platforms have been widely used for training in many applications, many of them still require the presence of human experts so that real-time feedback can be provided during learning to ensure that relevant skills are learned. For example, in a cochlear implant surgery facilitated by the surgical simulator in Figure 2.10, drilling requires professional navigation around sensitive anatomical structures such as the facial nerve, damage to which can cause permanent impairment such as facial paralysis [236, 238]. This has been one of the obstacles to the spread of SBL systems as a new type of learning platforms. As such, a number of works have been proposed to automate the generation of real-time feedback in SBL.

2.3.3 Feedback generation in simulation-based learning

SBL often works with multivariate time series data as a SBL task often consists of a series of steps over a period of time. As such, user skills are usually defined over a period of time, based on the values of certain skill metrics. In general, skill metrics in a typical SBL system include: 1) motion-based metrics, 2) time-based metrics, 3) position-based metrics, or 4) system settings. Motion-based metrics are often signals captured from haptic devices or sensors, for example in a driving simulator, the moving speed of the car and the engine rpm. Time-based metrics measure time-related quantities such as the duration of a user behavior and the reaction time to a circumstance change in military training. Position-based metrics record the spatial location of user operations with quantities such as position coordinates and distances to landmarks. System settings refer to settings that affect the learning environment or the quality of learning such as the magnification level in surgical simulation and the size of the drill applied to a surgical drilling. A more formal statement about the definition of user skill is as follows:

**Definition 1.** In SBL, user skill is a feature vector summarizing user behaviour over an arbitrary period of time and annotated with class labels.

For example, in a SBL environment for surgery, the level of skill can be defined by the quality of a stroke, a continuous motion of the drill with no abrupt
changes in direction. The period of time, over which the user behaviour is summarized, is the time interval of a complete stroke. Metrics that define the quality of a stroke within this time frame include measures such as stroke length, drilling speed, drilling acceleration, drilling duration, drill orientation/straightness, and drilling force [238]. We denote an individual skill metric as a feature, a vector of feature values that defines a user behavior is an instance and an instance is associated with a class label to denote its skill level (‘expert’ or ‘novice’).

From a technical perspective, the provision of feedback in SBL is a two-step process: 1) training a classifier to predict skill level from a multivariate stream of data, and 2) generating feedback that can change a skill vector from ‘novice’ to ‘expert’ with respect to the classifier [235, 237, 238]. In the first step, a prediction model has to be learned via supervised learning based on labelled stream data samples. The second step of feedback generation is an optimization process to find optimal transitions from novice class to expert class based on the prediction model under limited cost [236].

Figure 2.11 illustrates a typical real-time feedback process in SBL. It operates in two steps: 1) offline training and 2) real-time feedback provision [236]. In the offline stage, a feedback generation method is trained via supervised learning on labelled (typically novice/expert) skill samples. In real-time, when a trainee is practising on the simulator, novice skill will be captured and input into the feedback generation method to obtain feedback about where improvement is required. Technically, feedback is the suggested action that can improve novice skill to expert skill. Finally, the feedback will be delivered immediately back to the trainee to improve behaviour. Figure 2.12 shows an example of how generated feedback are delivered to the trainee in the form of text highlight and audio, in a virtual reality based surgical platform (see Figure 2.10 for more details of the platform itself). A survey on feedback delivering techniques can be found in [222].

When compared to applications such as self-driving cars and reinforcement learning agents as mentioned in Section 2.3.1, SBL focuses more on educational gains such as the acquisition of proper skills [105, 189]. As such, SBL requires a higher degree of “hands-on” experiential interaction. Rule-based feedback tutoring methods that work in domains such as algebra and physics are not
2. BACKGROUND

There are three major challenges for the generation of real-time feedback in SBL. First, generated feedback should be highly effective. That is, it should assist novices in gaining expertise. Second, feedback should be provided in real-time. Autonomous driving systems and reinforcement learning systems mainly focus on the outcomes and mostly deal with cognitive tasks. Therefore, feedback generation methods in these systems are not directly transferable to SBL, especially for non-cognitive SBL scenarios.
2.3. FEEDBACK GENERATION AND SIMULATION-BASED LEARNING

time (typically within 1 second) when novice behaviour is detected. This efficiency requirement is based on the average duration of a user movement, otherwise the trainee might already move on to the next move [166]. Delayed feedback can lead to confusion or even cause fatal consequences in reality. Third, when presenting feedback, only one aspect of performance/skill can be delivered at a time as practically, people can only focus on one thing at a time. This reduces distractions to the trainee and decreases cognitive load [199], thus increasing the usefulness of the feedback.

Although feedback generation in SBL is a rather new topic, there exist several notable works. The simplest way to generate feedback is the rule-based approaches. The “follow-me” method (ghost drill) [164] and the “step-by-step” method [224] are examples of this approach. The “follow-me” approach was proposed in [164] to guide the trainee by providing a ghost drill in a dental surgery simulation. The ghost feedback is a pre-recorded video of a human expert performing the same task. Wijewickrema et al. [224, 223] proposed to use a set of fixed rules defined by human experts to assess the skill level of trainees, and to generate feedback to encourage trainees to follow the fixed rules. However, it may be hard for a novice who has limited experience to follow a ghost drill at his own pace, and step-by-step feedback will not respond if the trainee does not follow the suggested paths.

Forestier et al. [48] applied Dynamic Time Warping (DTW) to classify a set of surgical processes performed by experts and novices in a surgical simulator. The DTW distances between a given surgical process and these novice and expert processes were computed. A new performed surgical process will be classified as novice if its closest process belonged to the novice group and vice versa. Experimental results indicated that DTW can be used to discriminate the surgical process as well as the level of expertise. However, the efficiency of DTW is susceptible to the time window size and the detection accuracy cannot be guaranteed at the beginning of a process where only a few strokes are available for detection.

A pattern-based model (EP) was recommended by Zhou et al. [236] to extract technical feedback in a surgical simulation, based on the Emerging Pattern algorithm [41]. Technical patterns were derived based on motion-based met-
Table 2.1: Summary of existing feedback generation methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Real-time</th>
<th>High Effectiveness</th>
<th>High Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yang et al. [230]</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Cui et al. [31]</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Rhienmora et al. [164]</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Wijewickrema et al. [224]</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Forestier et al. [48]</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Zhou et al. [236]</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Zhou et al. [237]</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

ricrs to represent expert and novice drilling manoeuvres separately. This was achieved by mining itemsets whose supports increased significantly from expert to novice group. Expert patterns were used to provide instructive feedback in real-time when novice patterns were identified. Patterns generated via this approach are easily interpretable as drilling styles. However, this approach suffers from low accuracy in novice detection as the pattern mining causes significant information loss.

The other approach suggested by Zhou et al. [236] is a RF model based greedy method. A RF prediction model was first trained on a set of pre-collected expert and novice instances, then used to detect novice skills in real-time when a trainee is practising. When a novice instance is detected, this method first searches its nearest expert instance from existing samples. Then, it starts a voting mechanism based on split features, only within those trees of the RF model that can correctly classify both the novice and expert instances. The first split feature where the two instances start to fall into different branches (along with the direction of the branch, positive or negative) receives a vote. The feature (and corresponding direction) with most votes will be chosen as the feedback. For the purpose of narration, we denote this method as Split Voting (SV). Although feedback generated by the SV method outperformed the compared baselines, the quality of the feedback is limited by existing expert samples.

The above reviewed feedback generation methods are summarized in Table
in terms of real-time suitability, effectiveness and generation efficiency. In this thesis, we propose the novel use of adversarial techniques for the generation of feedback in SBL. In Chapter 4, we develop feedback generation methods with ANNs and adversarial techniques to generate simple yet effective feedback as required in SBL. In Chapter 5, we further investigate efficient RF-based generation methods for feedback provision in SBL. We demonstrate, in a virtual reality based surgical platform, that our proposed methods are highly efficient and effective to be used for real-time feedback generation.

In this section, we have reviewed the problem of feedback generation, the background of SBL, and the application of feedback generation in SBL. We discussed the progress that have been achieved in the literature and limitations of existing works. In this thesis, we address these limitations by exploring the possibilities of applying adversarial techniques (discussed in Section 2.2.2) to feedback generation, which motivates two novel feedback generation methods based on ANN and RF models.

2.4 Noisy labels and deep neural networks

In this section, we discuss existing literature on noisy label based learning from two aspects: 1) traditional learning with noisy labels and 2) deep learning with noisy labels. We begin with a brief discussion of the noisy label problem in supervised learning, then review existing works from the above two aspects. In this thesis (Chapter 6), we investigate the learning behaviors of DNNs in the presence of noisy labels, and propose a novel learning strategy for training accurate DNNs against noisy labels.

2.4.1 Noisy labels

Training samples whose classes were incorrectly labeled are noisy samples and their labels are noisy labels. As discussed in Section 2.1.1 supervised learning requires the availability of annotated labels (discrete or continuous) in the training data for model learning. The quality of labels directly affect the quality of the model that can be learned from the data [139, 140]. Traditional training of
2. BACKGROUND

classifiers assumes a perfectly labelled training set [49, 51]. For example, the success of DNNs [101] is highly tied to the availability of large-scale annotated datasets, e.g. ImageNet [37]. However, data collection and labeling is often time-consuming and error-prone, where noisy labels can be easily introduced to contaminate training data. Even high quality datasets are tend to contain noisy labels (or label noise). For instance, even human professionals make 5% mistakes in labelling natural images [69].

Meanwhile, the growing computational power of hardware allows the design and training of very complex machine learning models such as DNNs with thousands of layers and millions of parameters [70, 201]. Such complex models usually require large-scale training data to be available for training [186]. However, collecting and annotating large-scale datasets is a costly and noisy process. And in many applications, large-scale annotated data with clean labels are not always obtainable, as it may takes enormous time and effort for human experts to label such data.

There are several cheap but noisy approaches to obtain large-scale data. For example, web searching, a cheap way to search massive data from the web such as web pages, online social networks and searching engines [26, 40]. Given a set of query keywords, web searching can easily retrieve a huge amount of data with some data may not be relevant. A fine-grained labeling is often required on webly searched data, however, this may take tremendous efforts to accomplish. A common approach is to label webly searched data by the query keywords or tags from surrounding texts. This approach can be very noisy depends on the quality of the keywords, for example, ‘apple’ can be a type of fruit or a company [213]. Therefore, webly searched data often contains a large proportion of noisy labels [26].

Another cheap but imperfect surrogate for labelling large-scale data is the “crowd-sourcing”. The labeling task is breakdown into smaller tasks that can be put up online through a cloud service to allow random users to label the data. For example, the ImageNet dataset, which consists of 15 million images of roughly 22,000 classes, was labeled via crowd-sourcing using the Amazon’s Mechanical Turk tool [101, 176]. Although crowd-sourcing provides the possibility to label large-scale data, it inevitably introduces noisy (incorrect) labels
Table 2.2: Types of labels for a ‘jasmine-cat’ dataset.

<table>
<thead>
<tr>
<th>true ‘jasmine’</th>
<th>labeled as ‘jasmine’</th>
<th>labeled as ‘cat’</th>
</tr>
</thead>
<tbody>
<tr>
<td>true ‘cat’</td>
<td>closed-set</td>
<td>clean</td>
</tr>
<tr>
<td>other class images</td>
<td>open-set</td>
<td>open-set</td>
</tr>
</tbody>
</table>

into the data, as web users are not properly trained to perform such a labelling task [213].

Noisy labels can be categorized into two types based on whether or not the underlying true classes of noisy samples are contained in the predefined class-set: 1) closed-set noisy labels and 2) open-set noisy labels. More specifically, a closed-set noisy label occurs when a noisy sample possesses a true class that is contained within the set of known classes in the training data. While, an open-set noisy label occurs when a noisy sample possesses a true class that is not contained within the set of known classes in the training data [213].

Table 2.2 demonstrates the relationship between clean labels and the two types of noisy labels in an exemplary ‘jasmine-cat’ scenario. The same example is illustrated in Figure 2.13. In closed-set noisy labels, cat and jasmine are mislabeled from one category to the other, but the true labels of these images are still cat or jasmine. For open-set noisy labels, there are some images labeled as cat or jasmine, but their true labels are neither cat nor jasmine, for instance, the zoo map and cartoon character in Figure 2.13. While most of existing works in the literature focus on the closed-set noisy label setting [90, 104, 140, 157, 163, 193, 208], recent works in deep learning start to investigate the open-set noisy labels [27, 213, 234]. In this thesis, we explore DNN learning behaviors in the closed-set setting.

Noisy label based learning is important for robust supervised learning. The presence of noisy labels in the training data will adversely affect model learning, thus lead to poor representation learning and prediction/generalization performance [142]. An extreme case is that the training labels are all incorrect, then the learned model is almost impossible to make any reliable predictions [233]. Although some models such as CNNs are robust to small proportions of noisy labels [4, 233], supervised learning models in general are vulnerable to
noisy labels [140, 194]. The goal of noisy label based learning is to train accurate prediction models when the labels used for training are noisy. In the next two subsections, we will review existing works in the literature: traditional learning with noisy labels and deep learning with noisy labels.

### 2.4.2 Traditional learning with noisy labels

A simple approach to handle noisy labels is to remove samples with suspicious labels from the training data via data cleansing [18]. This can be done by applying some outlier [79, 173] or anomaly detection [23] methods on the training samples, so that samples with detection scores above certain threshold will be determined to be noisy and can be removed from the training data [195]. Suspicous samples can also be identified based on their predictions from a classifier, for example, misclassified samples can be regarded as samples associated with noisy labels [88]. These methods often suffer from low detection accuracy which may lead to misdetection at significant levels [210].

Ensembles or iterative cleansing methods have been proposed to give relatively better detection performance [19, 91, 210]. For example, a sample will be identified as noisy if an ensemble of classifiers agree to remove it [210]. Near-
2.4. NOISY LABELS AND DEEP NEURAL NETWORKS

Est neighbour approaches have also been used to remove noisy samples, which reduce samples that have limited influence on their neighbours [94, 210]. Clustering methods were also suggested to identify noisy label samples, under the assumption that clustered samples should have the same labels otherwise noisy [15, 231]. However, these cleansing methods are often challenged by the difficulty in distinguishing samples that are inherently hard to learn from those with noisy labels [67]. Moreover, removing samples from training data may lead to insufficient data remained for model learning.

An alternative approach is to model the label noise explicitly into a noise-tolerant learning algorithm for probabilistic machine learning models. Probabilistic noisy label modeling is one such type of methods. The assumption is that the true labels $Y$ are drawn from a prior distribution $P_Y$, the samples are drawn from the conditional distribution $P_{X|Y}$, and the observed labels $\hat{Y}$ are drawn from the conditional distribution $P_{\hat{Y}|Y}$. The samples and observed labels are known while the hidden true labels have to be learned or inferred from the data [106]. Probabilistic modeling with Bayesian priors such as Beta priors [53, 92] and Dirichlet priors [121, 174] have been proposed for noisy label learning [5, 198]. For example, Bayesian priors were considered in training logistic regression [1, 33, 56, 132], hidden Markov models [55] and graphical models [95] on noisy labeled data. Generative probabilistic models have also been investigated for noisy label learning with Gaussian [106] or non-Gaussian conditional class distributions [118] and multi-class problems [13].

Model-based noise-tolerant methods have also been proposed for learning robust perceptron algorithms with noisy labels. This includes some weighting or correction vector to prevent instances from taking too large weights [96, 99, 116]. Bylander [20] investigated the learning of linear threshold functions such as perceptron [170] against uniform label noise, while Li and Long [116] designed an incremental algorithm for training linear threshold functions with large margins, which demonstrated robustness against random label noise [52]. Krauth et al. [159] studied the stability of spin-glass-like neural networks against random uncorrelated patterns, and provided optimal stability learning rules for robust learning with noisy labels [99]. Noise tolerant variants of perceptron were proposed in [96], for example, perceptron ensembles, perceptron with
2. BACKGROUND

large margins or soft-margins. Stempfel and Ralaivola [190] proposed a noise tolerant perceptron learning algorithm with random or deterministic projections to learn non-linearly separable concepts in the presence of uniform label noise.

Other noise-tolerant learning approaches have also been proposed to handle noisy labels. For example, the use of belief functions [38, 39] to allow the modelling of label confidence, and the beliefs can be inferred from the data [231]. Weighting strategies, where mislabeled samples will receive smaller weights than correctly labeled samples, have been used to train non-probabilistic models such as SVMs [54, 119] and boosting-based models such as Adaboost [35, 52, 89, 125, 239]. Adjusted loss functions that are less sensitive to noisy labels can also be used for training non-probabilistic models [98, 130, 191]. A comprehensive survey on traditional noise label based learning can be found in [50, 142].

2.4.3 Deep learning with noisy labels

Exploring and understanding the generalization behavior of deep networks under the noisy label setting has recently attracted significant attention. Zhang et al. [233] showed that deep networks are capable of memorizing completely random labels, and models learned with noisy labels generalize poorly. They argued that deep networks employ case-by-case memorization on training samples and labels in this scenario. Krueger et al. [102] later highlighted that DNNs exhibit different learning styles on datasets with clean labels versus those on datasets with either noisy inputs or noisy labels. They found that DNNs require more capacity, longer training time to fit noisy labels and the learned hypotheses are more complex.

Arpit et al. [4] further identified the two stages of learning of deep nets with noisy labels: an early stage of simple pattern learning and refining, and a later stage of label memorization. They also showed that dropout regularization can mitigate overfitting to noisy labels. Shwartz-Ziv and Tishby [184] demonstrated that, on data with clean labels, deep nets with tanh activations undergo an initial label fitting phase, and a subsequent compression phase. They also argued that information compression is related to the excellent generalization
performance of deep nets. However, Michael et al. [178] conducted experiments where information compression was not found to occur for ReLU [59] networks. Despite of these works, a comprehensive understanding of DNN learning behaviors with noisy labels, and a practical measurement that can determine whether or not a learning process is noisy is still unestablished, which we will address in this paper.

A number of strategies have been proposed to address the problem of learning accurate DNNs in the presence of noisy labels. One strategy is to explicitly or implicitly formulate the noise model and use a corresponding noise-aware approach. Symmetric label noise that is independent of the true label was modeled in [104], and asymmetric label noise that is conditionally independent of individual samples was modeled in [140] [193]. There are also more complex noise models where true labels and noisy labels can be characterized by directed graphical models [226], conditional random fields [208], neural networks [209] or knowledge graphs [117]. These methods aim to correct noisy labels to their true labels via a clean label inference step, or assign smaller weights to those noisy samples. For the modeling of label noise, these methods often require an expensive detection process, or an extra dataset to be available with noisy labels and their ground truth labels. They may also rely on specific assumptions about the noise model.

Other approaches utilize correction methods to adjust the loss function to reduce the influence of noisy samples on model learning. Backward [157] and Forward [157] are two such correction methods that use an estimated or learned factor to modify the loss function. [60] [194] further augmented the correction architecture by adding a linear layer on top of the network. Bootstrap [163] is another loss correction method that replaces the target labels used by a loss function with a combination of the raw labels and their predicted labels.

Reweighting-based regularization techniques have also been suggested to train DNNs in the presence of noisy labels. Liu and Tao [122] proposed the use of reweighted loss function via importance reweighting to decrease the importance of noisy samples based on estimated probabilities of the samples being noisy. Jiang et al. [90] proposed to train a second neural network called the “MentorNet” to automatically learn the importance of each sample, and use the
learned importance to reweight training samples for training the target network — the “StudentNet”. The two networks are trained in parallel to learn a more regularized hypothesis that is robust to noisy labels.

As pointed out by Wang et al. [213], noisy labels can be closed-set or open-set. Closed-set noisy labels means that their true classes are meaningful as they still belong to a known class-set, however, the true classes of open-set noisy labels are not meaningful and can be arbitrary classes that are irrelevant to the learning task. The open-set noisy label problem is a rather new topic arises with deep learning and large-scale web data [213]. They proposed an iterative learning framework with three modules: a noisy label detection module, a reweighting module and a contrastive learning module. The noisy labels were detected and reweighted during the learning process, and their representations were pulled away from clean samples by contrastive learning with Siamese network. The proposed learning framework has been proved to be able to deal with closed-set as well as open-set noisy labels.

While existing works have studied the differences between learning with clean versus noisy labels, a full picture of DNN learning behavior and its implication for DNN generalization is yet to emerge. In this thesis, we explore the learning behaviors of deep networks in the presence of noisy labels, and provide a new perspective of understanding to the community based on dimensionality-based analysis on the deep representation space of deep networks. Moreover, we propose a novel training strategy to train robust DNNs against large proportions of noisy labels.

In this section, we have reviewed the problem of learning with noisy labels in the supervised context, and discussed existing works on understanding DNN learning behavior and traditional/deep learning with noisy labels. In this thesis, we investigate learning with noisy labels in the context of deep learning and deep classification networks. Different to existing works, we link noisy label based learning with DNN understanding. In particular, we use random noisy labels to explore the different learning behaviours of DNNs on clean versus noisy labels. Our study adds another perspective into DNN understanding based on dimensional characterization of DNN representation space, and shows how this can lead to the development of an effective learning strategy.
Chapter 3

Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality

We first study the vulnerability of deep networks (or DNNs) to adversarial perturbations. DNNs have recently been shown to be vulnerable to adversarial examples, which are carefully crafted instances by adversarial perturbations that can mislead DNNs to make errors during prediction. To better understand such attacks, a characterization is needed of the properties of regions (the so-called “adversarial subspaces”) in which adversarial examples lie. We tackle this challenge by characterizing the dimensional properties of adversarial regions, via the use of Local Intrinsic Dimensionality (LID). LID assesses the space-filling capability of the region surrounding a reference example, based on the distance distribution of the example to its neighbors. We first provide explanations about how adversarial perturbation can affect the LID characteristic of adversarial regions, and then show empirically that LID characteristics can facilitate the distinction of adversarial examples generated using state-of-the-art attacks. As a proof-of-concept, we show that a potential application of LID is to distinguish adversarial examples, and the preliminary results show that it can outperform several state-of-the-art detection measures by large margins for five
3. CHARACTERIZING ADVERSARIAL SUBSPACES USING LOCAL INTRINSIC DIMENSIONALITY

attack strategies considered in this chapter across three benchmark datasets.¹

3.1 Introduction

DNNs are highly expressive models that have achieved state-of-the-art performance on a wide range of complex problems, such as speech recognition [74] and image classification [101]. However, recent studies have found that DNNs can be compromised by adversarial examples [63, 202, 145]. These intentionally-perturbed inputs can induce the network to make incorrect predictions at test time with high confidence, even when the examples are generated using different networks [22, 124, 151]. The amount of perturbation required is often small, and (in the case of images) imperceptible to human observers. This undesirable property of deep networks has become a major security concern in real-world applications of DNNs, such as self-driving cars and identity recognition [45, 183]. In this chapter, we aim to further understand adversarial attacks by characterizing the regions within which adversarial examples reside.

Each adversarial example can be regarded as being surrounded by a connected region of the domain (the “adversarial region” or “adversarial subspace”) within which all points subvert the classifier in a similar way. Adversarial regions can be defined not only in the input space, but also with respect to the activation space of different DNN layers [202]. Developing an understanding of the properties of adversarial subspaces is a key requirement for adversarial defense. Under the assumption that data can be modeled in terms of collections of manifolds, several works have attempted to characterize the properties of adversarial subspaces, but no definitive method yet exists which can reliably discriminate adversarial regions from those in which normal data can be found. [202] argued that adversarial subspaces are low probability regions (not naturally occurring) that are densely scattered in the high dimensional representation space of DNNs. However, a linear formulation argues that adversarial sub-

3.1. INTRODUCTION

spaces span a contiguous multidimensional space, rather than being scattered randomly in small pockets \[63, 215, 205\]. \[205\] further emphasize that adversarial subspaces lie close to (but not on) the data submanifold. Similarly, it has also been found that the boundaries of adversarial subspaces are close to legitimate data points in adversarial directions, and that the higher the number of orthogonal adversarial directions of these subspaces, the more transferable they are to other models \[207\]. To summarize, with respect to the manifold model of data, the known properties of adversarial subspaces are: (1) they are of low probability, (2) they span a contiguous multidimensional space, (3) they lie off (but are close to) the data submanifold, and (4) they have class distributions that differ from that of their closest data submanifold.

Among adversarial defense/detection techniques, Kernel Density (KD, see Equation (2.23)) estimation has been proposed as a measure to identify adversarial subspaces \[47\]. \[21\] demonstrated the usefulness of KD-based detection, taking advantage of the low probability density generally associated with adversarial subspaces. However, in this chapter we will show that kernel density is not effective for the detection of some forms of attack. In addition to kernel density, there are other density-based measures, such as the number of nearest neighbors within a fixed distance, and the mean distance to the \(k\) nearest neighbors (\(k\)-mean distance). Again, these measures have limitations for the characterization of local adversarial regions. For example, in Figure 3.1 the three density measures fail to differentiate an adversarial example (red star) from a normal example (black cross), as the two examples are locally surrounded by the same number of neighbors (50), and have the same \(k\)-mean distance (KM=0.19) and kernel density (KD=0.92).

As an alternative to density measures, Figure 3.1 leads us to consider expansion-based measures of intrinsic dimensionality as a potentially effective method of characterizing adversarial examples. Expansion models of dimensionality assess the local dimensional structure of the data — such models have been successfully employed in a wide range of applications, such as manifold learning, dimension reduction, similarity search, correlation analysis and anomaly detection \[3, 81, 167, 214\]. Although earlier expansion models characterize intrinsic dimensionality as a property of data sets, the Local Intrinsic Dimensionality (LID)
3. CHARACTERIZING ADVERSARIAL SUBSPACES USING LOCAL INTRINSIC DIMENSIONALITY

Figure 3.1: This example shows how density measures can fail to characterize the spatial properties of adversarial regions. The Gaussian kernel with bandwidth 0.2 is used for KD.

fully generalizes this concept to the local distance distribution from a reference point to its neighbors — the dimensionality of the local data submanifold in the vicinity of the reference point is revealed by the growth characteristics of the cumulative distribution function. In this chapter, we use LID to characterize the intrinsic dimensionality of adversarial regions, and attempt to test how well the estimates of LID can be used to distinguish adversarial examples. Note that the main goal of LID is to characterize properties of adversarial examples, instead of being applied as a pure defense method, which requires stronger assumptions on the current threat model. In Figure 3.1, the estimated LID of the adversarial example (LID \( \approx 4.36 \)) is much higher than that of the referenced normal data sample (LID \( \approx 1.53 \)), illustrating that the estimated LID can efficiently capture the intrinsic dimensional properties of adversarial regions.

3.2 Local Intrinsic Dimensionality

In the theory of intrinsic dimensionality, classical expansion models (such as the expansion dimension and generalized expansion dimension) measure the rate of growth in the number of data objects encountered as the distance from the reference sample increases. As an intuitive example, in Euclidean
space, the volume of an \( m \)-dimensional ball grows proportionally to \( r^m \), when its size is scaled by a factor of \( r \). From this rate of volume growth with distance, the expansion dimension \( m \) can be deduced as:

\[
\frac{V_2}{V_1} = \left( \frac{r_2}{r_1} \right)^m \Rightarrow m = \frac{\ln(V_2/V_1)}{\ln(r_2/r_1)}.
\] (3.1)

By treating probability mass as a proxy for volume, classical expansion models provide a local view of the dimensional structure of the data, as their estimation is restricted to a neighborhood around the sample of interest. Transferring the concept of expansion dimension to the statistical setting of continuous distance distributions leads to the formal definition of LID [81].

### 3.2.1 Definition of LID

**Definition 2 (Local Intrinsic Dimensionality).**

Given a data sample \( x \in X \), let \( R > 0 \) be a random variable denoting the distance from \( x \) to other data samples. If the cumulative distribution function \( F(r) \) of \( R \) is positive and continuously differentiable at distance \( r > 0 \), the LID of \( x \) at distance \( r \) is given by:

\[
\text{LID}_F(r) \triangleq \lim_{\epsilon \to 0} \frac{\ln \left( F((1 + \epsilon) \cdot r)/F(r) \right)}{\ln(1 + \epsilon)} = \frac{r \cdot F'(r)}{F(r)},
\] (3.2)

whenever the limit exists.

\( F(r) \) is analogous to the volume \( V \) in Equation (3.1); however, we note that the underlying distance measure need not be Euclidean. The last equality of Equation (3.2) follows by applying L’Hôpital’s rule to the limits [81]. The local intrinsic dimension at \( x \) is in turn defined as the limit, when the radius \( r \) tends to zero:

\[
\text{LID}_F = \lim_{r \to 0} \text{LID}_F(r).
\] (3.3)

\( \text{LID}_F \) describes the relative rate at which its cumulative distance function \( F(r) \) increases as the distance \( r \) increases from 0, and can be estimated using the distances of \( x \) to its \( k \) nearest neighbors within the sample [3].
In the ideal case where the data in the vicinity of \( x \) is distributed uniformly within a submanifold, \( \text{LID}_F \) equals the dimension of the submanifold; however, in general these distributions are not ideal, the manifold model of data does not perfectly apply, and \( \text{LID}_F \) is not an integer. Nevertheless, the local intrinsic dimensionality does give a rough indication of the dimension of the submanifold containing \( x \) that would best fit the data distribution in the vicinity of \( x \). We refer readers to [81, 82] for more details concerning the LID model.

### 3.2.2 Estimation of LID

According to the branch of statistics known as extreme value theory, the smallest \( k \) nearest neighbor distances could be regarded as extreme events associated with the lower tail of the underlying distance distribution. Under very reasonable assumptions, the tails of continuous probability distributions converge to the Generalized Pareto Distribution (GPD), a form of power-law distribution [29]. From this, [3] developed several estimators of LID to heuristically approximate the true underlying distance distribution by a transformed GPD; among these, the Maximum Likelihood Estimator (MLE) exhibited a useful trade-off between statistical efficiency and complexity. Given a reference sample \( x \sim \mathcal{P} \), where \( \mathcal{P} \) represents the data distribution, the MLE estimator of the LID at \( x \) is defined as follows:

\[
\hat{\text{LID}}(x) = -\left(\frac{1}{k} \sum_{i=1}^{k} \log \frac{r_i(x)}{r_k(x)}\right)^{-1}.
\]

(3.4)

Here, \( r_i(x) \) denotes the distance between \( x \) and its \( i \)-th nearest neighbor within a sample of points drawn from \( \mathcal{P} \), where \( r_k(x) \) is the maximum of the neighbor distances. In practice, the sample set is drawn uniformly from the available training data (omitting \( x \) itself), which itself is presumed to have been randomly drawn from \( \mathcal{P} \). We emphasize that the LID defined in Equation (3.3) is a theoretical quantity, and that \( \hat{\text{LID}} \) as defined in Equation (3.4) is its estimate. In the remainder of this chapter, we will refer to Equation (3.4) to calculate LID estimates.
3.3 Characterizing adversarial regions

Our aim is to gain a better understanding of adversarial regions, and thereby derive potential defenses and provide new directions for more efficient attacks. We begin by providing some motivation with respect to the manifold model of data as to how adversarial perturbation might affect the LID characteristic of adversarial regions. We then show how a detector can potentially be designed using LID estimates to discriminate between adversarial and normal examples.

3.3.1 LID of adversarial subspaces

Consider a sample $x \in X$ lying within a data submanifold $S$, where $x$ is a randomly sampled dataset from $\mathcal{P}$ consisting only of normal (unperturbed) examples. Adversarial perturbation of $x$ typically results in a new sample $x'$ whose coordinates differ from those of $x$ by very small amounts. Assuming that $x'$ is indeed a successful adversarial perturbation of $x$, the theoretical LID value associated with $x$ is simply the dimension of $S$, whereas the theoretical LID value associated with $x'$ is the dimension of the adversarial subspace within which it resides. Recent work in [2] shows that the magnitude of the perturbation required to make changes in the expected nearest neighbor ranking tends to zero as the LID and the data sample size tend to infinity.

Since perturbation schemes generally allow the modification of all data coordinates, they exploit the full degrees of freedom afforded by the representational dimension of the data domain. As pointed out by [63, 215, 205], $x'$ is very likely to lie outside $S$ (but very close to $S$ — in a high-dimensional contiguous space). In applications involving high-dimensional data, the representational dimension is typically far larger than the intrinsic dimension of any given data submanifold, which implies that the theoretical LID of $x'$ is far greater than that of $x$.

In practice, however, the values of LID must be estimated from local data samples. This is typically done by applying an appropriate estimator (such as the MLE estimator shown in Equation (3.4)) to a $k$-nearest neighborhood of the test samples, for some appropriate fixed choice of $k$. Typically, $k$ is chosen large.
enough for the estimation to stabilize, but not so large that the sample is no 
longer local to the test sample. If the dimension of $S$ is reasonably low, one can 
expect the estimation of the LID of $x$ to be reasonably accurate.

For the adversarial subspace, the samples appearing in the neighborhood of 
$x'$ can be expected to be drawn from more than one manifold. The proximity 
of $x'$ to $S$ means that the neighborhood is likely to contain neighbors lying in $S$; 
however, if the neighborhood were composed mostly of samples drawn from $S$, 
$x'$ would not likely be an adversarial example. Thus, the neighbors of $x'$ taken 
together are likely to span a subspace of intrinsic dimensionality much higher 
than any of these submanifolds considered individually, and the LID estimate 
computed for $x'$ can be expected to reveal this.

### 3.3.2 Efficiency through minibatch sampling

Computing neighborhoods with respect to the entirety of the dataset $x$ can be 
prohibitively expensive, particularly when the (global) intrinsic dimension-
ality of $x$ is too high to support efficient indexing. For this reason, when $x$ is 
large, the computational cost can be reduced by estimating the LID of an ad-
versarial example $x'$ from its $k$-nearest neighbor set within a randomly-selected 
sample (minibatch) of the dataset $x$. Since the LID estimation model regards the 
distances from $x'$ to the members of $x$ as determined by independently-drawn 
samples from a distribution $P$, the estimator can also be applied to the distances 
induced by any random minibatch, as it too would be drawn independently 
from the same distribution $P$.

Provided that the minibatch is chosen sufficiently large so as to ensure that 
the $k$-nearest neighbor sets remain in the vicinity of $x'$, estimates of LID com-
puted for $x'$ within the minibatch would resemble those computed within the 
full dataset $x$. Conversely, as the size of the minibatch is reduced, the variance 
of the estimates would increase. However, if the gap between the true LID val-
ues of $x$ and $x'$ is sufficiently large, even an extremely small minibatch size and 
/ or small neighborhood size could conceivably produce estimates whose dif-
fERENCE is sufficient to reveal the adversarial nature of $x'$. As we shall show 
in Section 3.4.2 discrimination between adversarial and non-adversarial exam-
3.3. CHARACTERIZING ADVERSARIAL REGIONS

Examples turns out to be possible even for minibatch sizes as small as 100, and for neighborhood sizes as small as 20.

3.3.3 Using LID to characterize adversarial examples

We next describe how LID estimates can serve as features to train a detector to distinguish adversarial examples. Note that here we only aim to train a baseline classifier to demonstrate how well LID can characterize adversarial examples. Robust detection taking different attack variations into account, such as attack confidence, will be left as future work. Our methodology requires that training sets be comprised of three types of examples: adversarial, normal and noisy. This replicates the methodology used in [47, 21], where the rationale for including noisy examples is that DNNs are required to be robust to random input noise [46] and noisy inputs should not be identified as adversarial attacks. A classifier can be trained by using the training data to construct features for each sample, based on its LID within a minibatch of samples across different layers, where the class label is assigned positive for adversarial examples and assigned negative for normal and noisy examples.

Algorithm 3.1 describes how the LID features can be extracted for training an LID-based classifier. Given an initial training dataset and a DNN pre-trained on the initial training dataset, the algorithm outputs a classifier trained using LID features. As in previous studies [21, 47], we assume that the initial training dataset is free of adversarial examples — that is, all examples in the dataset are considered ‘normal’ to begin with. The extraction of LID features first begins with the generation of adversarial and noisy counterparts to normal examples (step 3 and 4) in each minibatch. One minibatch of normal examples ($B_{norm}$) is used for generating 2 counterpart minibatches of examples: one adversarial ($B_{adv}$) and one noisy ($B_{noisy}$). The adversarial examples are generated using an adversarial attack on normal examples (step 3), while noisy examples are generated by adding random noise to normal examples, subject to the constraint that the magnitude of perturbation undergone by a noisy example is the same as the magnitude of perturbation undergone by its counterpart adversarial example (step 4). One minibatch of normal examples is converted to an equal number of
Algorithm 3.1 Training phase for LID-based adversarial classifier

Input:
- $X$: a dataset of normal examples
- $H(x)$: a pre-trained DNN with $L$ transformation layers
- $k$: the number of nearest neighbors for LID estimation

Output: Detector(LID) # a detector

1: $\operatorname{LID}_\text{neg} = [], \operatorname{LID}_\text{pos} = []$
2: for $B_{\text{norm}}$ in $X$ do # $B_{\text{norm}}$: a minibatch of normal examples
3: $B_{\text{adv}} :=$ adversarial attack $B_{\text{norm}}$ # $B_{\text{adv}}$: a minibatch of adversarial examples
4: $B_{\text{noisy}} :=$ add random noise to $B_{\text{norm}}$ # $B_{\text{noisy}}$: a minibatch of noisy examples
5: $N = |B_{\text{norm}}|$ # number of examples in $B_{\text{norm}}$
6: $\operatorname{LID}_\text{norm}, \operatorname{LID}_\text{noisy}, \operatorname{LID}_\text{noisy} = \text{zeros}[N, L]$
7: for $i$ in $[1, L]$ do # $i$-th layer activations of $B_{\text{norm}}$
8: $A_{\text{norm}} = H^i(B_{\text{norm}})$
9: $A_{\text{adv}} = H^i(B_{\text{adv}})$ # $i$-th layer activations of $B_{\text{adv}}$
10: $A_{\text{noisy}} = H^i(B_{\text{noisy}})$ # $i$-th layer activations of $B_{\text{noisy}}$
11: for $j$ in $[1, N]$ do # $i$-th nearest neighbor in $A_{\text{norm}}$
12: $\operatorname{LID}_\text{norm}[j, i] = -(\frac{1}{k} \sum_{i=1}^{k} \log \frac{r_i(A_{\text{norm}}[j], A_{\text{norm}})}{r_k(A_{\text{norm}}[j], A_{\text{norm}})})^{-1}$
13: $\operatorname{LID}_\text{adv}[j, i] = -(\frac{1}{k} \sum_{i=1}^{k} \log \frac{r_i(A_{\text{adv}}[j], A_{\text{adv}})}{r_k(A_{\text{adv}}[j], A_{\text{adv}})})^{-1}$
14: $\operatorname{LID}_\text{noisy}[j, i] = -(\frac{1}{k} \sum_{i=1}^{k} \log \frac{r_i(A_{\text{noisy}}[j], A_{\text{norm}})}{r_k(A_{\text{noisy}}[j], A_{\text{noisy}})})^{-1}$
15: $\triangleright r_i(A[j], A_{\text{norm}}):$ the $L_2$ distance of $A[j]$ to its $i$-th nearest neighbor
16: end for
17: end for
18: $\operatorname{LID}_\text{neg}.\text{append}(\operatorname{LID}_\text{norm}), \operatorname{LID}_\text{neg}.\text{append}(\operatorname{LID}_\text{noisy})$
19: $\operatorname{LID}_\text{pos}.\text{append}(\operatorname{LID}_\text{adv})$
20: end for
21: Detector(LID) = train a classifier on $(\operatorname{LID}_\text{neg}, \operatorname{LID}_\text{pos})$

adversarial examples after step 3, and an equal number of noisy examples after step 4.

The LID associated with each example (either normal, adversarial or noisy) is estimated from its $k$ nearest neighbors in the normal minibatch (steps 12-14), using Equation (3.4). For any new unknown test example, a minibatch consisting only of normal training examples is used to estimate LID. For each example and each transformation layer in the DNN, an LID estimate is calcu-
lated. The distance function needed for this estimate uses the activation values of the neurons in the given layer as inputs (step 7). As will be discussed in Section 3.4.2, we use all transformation layers, including conv2d, max-pooling, dropout, ReLU and softmax, since we expect adversarial regions to exist in each layer of the DNN representation space. The LID estimates associated with the example are then used as feature values (one feature for each transformation layer). Finally, a classifier (such as logistic regression) is trained using the LID features. Test examples can then be classified by the LID-based classifier to either the positive (adversarial) or negative (non-adversarial) class by means of its LID-based feature values.

### 3.4 Evaluating LID-based characterization of adversarial examples

In this section, we evaluate the discrimination power of LID-based characterization against five adversarial attack strategies — FGM, BIM-a, BIM-b, JSMA, and Opt, as introduced in Section 2.2.2. These attack strategies were selected for our experiments due to their reported effectiveness and their diversity. For each of the 5 forms of attack, the LID detector is compared with the state-of-the-art detection measures KD and BU (see Equation (2.24)) as discussed in Section 2.2.4, with respect to three benchmark image datasets: MNIST [109], CIFAR-10 [100] and SVHN [143]. Each of these three datasets is associated with a designated training set and test set. Before reporting and discussing the results, we first describe the experimental setup.

#### 3.4.1 Experimental setup

##### 3.4.1.1 Training and testing setup

For each of the three image datasets, a DNN classifier was independently pre-trained on its designated training set (the pre-train set), and its designated test set was used for testing (the pre-test set). Any pre-test images not correctly classified were discarded, and the remaining images were subdivided into train (80%)
and test (20%) sets for subsequent processing including training or evaluating detectors. This is to simulate the real-world situation where an adversary only can manipulate pre-test images. Both of these sets were randomly partitioned into minibatches of size 100, for later use in the computation of LID characteristics.

The LID-, KD- and BU-based detectors were trained separately on the train set using the scheme in Algorithm 3.1, with the calculation of LID estimates replaced by KD and BU calculation for their respective detectors. All three detectors were then evaluated against equal numbers of normal, noisy and adversarial images crafted from members of the test set, as described in Steps 2-4 of Algorithm 3.1. The LID, KD and BU characteristics of those test images were then generated as shown in Steps 1-19 of Algorithm 3.1. It should be noted that no images of the test set were examined during any of the training processes, so as to avoid cross contamination.

The adversarial examples for both training and testing were generated by applying one of the five selected attacks. The statistics of the five attacks on the three datasets is reported in Table 3.1. Following the procedure outlined in [47], the noisy examples for the JSMA attack were crafted by changing the values of a randomly-selected set of pixels to either their minimum or maximum (determined randomly), where the number of pixels to be adjusted was chosen to be equal to the number of pixels perturbed in the generation of adversarial examples. For the other attack strategies, $L_2$ Gaussian noise was added to the pixel values instead of setting them to their minimum or maximum. As suggested by [21, 47], we used the logistic regression classifier as detector, and report its AUC score as the metric for performance.

### 3.4.1.2 Deep neural networks for pretraining

The pretrained DNN used for MNIST was a 5-layer ConvNet with max-pooling and dropout. It achieved 99.29% classification accuracy on (normal) pre-test images. For CIFAR-10, a 12-layer ConvNet with max-pooling and dropout was used. This model reported an accuracy of 84.56% on (normal) pre-test images. For SVHN, we trained a 6-layer ConvNet with max-pooling and dropout. It
3.4. EVALUATING LID-BASED CHARACTERIZATION OF ADVERSARIAL EXAMPLES

Table 3.1: The $L_2$ mean perturbation and model accuracy (%) on adversarial examples.

<table>
<thead>
<tr>
<th></th>
<th>MNIST $L_2$</th>
<th>Acc.</th>
<th>CIFAR $L_2$</th>
<th>Acc.</th>
<th>SVHN $L_2$</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGM</td>
<td>6.26</td>
<td>11.09</td>
<td>2.74</td>
<td>3.15</td>
<td>7.09</td>
<td>6.17</td>
</tr>
<tr>
<td>BIM-a</td>
<td>2.30</td>
<td>10.43</td>
<td>0.48</td>
<td>0.00</td>
<td>0.83</td>
<td>0.13</td>
</tr>
<tr>
<td>BIM-b</td>
<td>5.42</td>
<td>10.42</td>
<td>3.39</td>
<td>0.00</td>
<td>5.53</td>
<td>0.13</td>
</tr>
<tr>
<td>JSMA</td>
<td>5.40</td>
<td>10.00</td>
<td>3.64</td>
<td>0.04</td>
<td>3.09</td>
<td>0.16</td>
</tr>
<tr>
<td>Opt</td>
<td>4.21</td>
<td>3.92</td>
<td>0.37</td>
<td>0.01</td>
<td>0.59</td>
<td>0.26</td>
</tr>
</tbody>
</table>

achieved 92.18% accuracy on (normal) pre-test images. We deliberately did not tune the DNNs, as their performance was close to the state-of-the-art and could thus be considered sufficient for use in an adversarial study [47].

3.4.1.3 Parameter tuning

We tuned the bandwidth ($\sigma$) parameter for KD, and the number of nearest neighbors ($k$) for LID, using nested cross validation within the training set ($\text{train}$). Using the AUC values of detection performance, the bandwidth was tuned using a grid search over the range $[0, 10]$ in log-space, and neighborhood size was tuned using a grid search over the range $[10, 100]$ with respect to a minibatch of size 100. For a given dataset, the parameter setting selected was the one with highest AUC averaged across all attacks. The optimal bandwidths chosen for MNIST, CIFAR-10 and SVHN were 3.79, 0.26, and 1.0, respectively, while the value of $k$ for LID estimation was set to 20 for MNIST and CIFAR-10, and 30 for SVHN. For BU, we chose the number of prediction runs to be $T = 50$ in all experiments. We did not tune this parameter, as it is not considered to be sensitive for choices of $T$ greater than 20 [21].

Our implementation is based on the detection framework of [47]. For FGM, JSMA, BIM-a, and BIM-b attack strategies, we used the cleverhans library [150], and for the Opt attack strategy, we used the author’s implementation [22]. We scaled all image feature values to the interval $[0, 1]$. 

67
3. CHARACTERIZING ADVERSARIAL SUBSPACES USING LOCAL INTRINSIC DIMENSIONALITY

3.4.2 LID characteristics of adversarial examples

We provide empirical results showing the LID characteristics of adversarial examples generated by Opt, the most effective of the known attack strategies. The left subfigure in Figure 3.2 shows the LID scores (at the softmax layer) of 100 randomly selected normal, noisy and adversarial (Opt) examples from the CIFAR-10 dataset. We observe that at this layer, the LID scores of adversarial examples are significantly higher than those of normal or noisy examples. This supports our expectation that adversarial regions have higher intrinsic dimensionality than normal data regions (as discussed in Section 3.3). It also suggests that the transition from normal example to adversarial example may follow directions in which the complexity of the local data submanifold significantly increases, leading to an increase in estimated LID values.

In the right subfigure of Figure 3.2, we further show that the LID scores of adversarial examples are more easily discriminated from those of other examples at deeper layers of the network. The 12-layer ConvNet used for CIFAR-10 consists of 26 transformation layers: the input layer ($L_0$), conv2d/max-pooling ($L_{1-17}$), dense/dropout ($L_{18-24}$) and the final softmax layer ($L_{25}$). The estimated LID characteristics of adversarial examples become distinguishable (detection AUC $> 0.5$) at the dense layers ($L_{18-24}$), and significantly different at the softmax layer ($L_{25}$). This suggests that the fully-connected and softmax transformations may be more sensitive to adversarial perturbations than convolutional transformations. Figure 3.3 illustrates LID characteristics of the most effective attack strategy known to date, Opt, on the MNIST and SVHN datasets. On both datasets, the LID scores of adversarial examples are significantly higher than those of normal or noisy examples. In the right-hand plot, the LID scores of normal examples and its noisy counterparts appear superimposed due to their similarities.

With regard to the stability of performance based on parameter variation ($k$ for LID, or bandwidth for KD), we can see from Figure 3.4 that LID is more stable than KD, exhibiting less variation in AUC as the parameter varies. From this figure, we also see that KD requires significantly different optimal settings for different types of data. For simpler datasets such as MNIST and SVHN, KD
3.4. EVALUATING LID-BASED CHARACTERIZATION OF ADVERSARIAL EXAMPLES

Figure 3.2: The left-hand figure shows the LID scores (at the softmax layer) of 100 normal (blue), noisy (green), and Opt attack (red x-cross) examples from the CIFAR-10 dataset. The scores have been scaled to the interval [0,1] using min-max normalization. The blue and green lines appear superimposed due to similarities in the LID scores for normal and noisy examples. The right-hand figure shows the detection performance (AUC) based on LID scores computed at different layers. $L_i$ denotes the $i$-th transformation layer.

Figure 3.3: The plots show the normalized LID scores of 100 randomly selected normal (blue), noisy (green) and Opt attack (red x-cross) examples. The noisy and adversarial examples were generated from the normal examples. The left-hand plot shows the scores (at the pre-softmax layer) of MNIST examples, while the right-hand plot shows LID scores (at the softmax layer) of SVHN examples. Normal and noisy example curves appear superimposed in the right-hand figure due to the similarity of their values.

requires quite high bandwidth choices for best performance.
3. CHARACTERIZING ADVERSARIAL SUBSPACES USING LOCAL INTRINSIC DIMENSIONALITY

Figure 3.4: Top row: tuning bandwidth $\sigma$ for KD using a grid search over the range $[0, 10]$ in log-space, separately for each dataset. Bottom row: tuning $k$ for LID using a grid search over the range $[10, 100]$ for minibatch size 100, separately for each dataset. The vertical dashed lines denote the selected parameter choice.

3.4.3 Analysis of LID properties

3.4.3.1 LID outperforms KD and BU

We compare the performance of LID-based detection with that of detectors trained with features of KD and BU individually, as well as a detector trained with a combination of KD and BU features (denoted as ‘KD+BU’). As shown in Table 3.2, LID outperforms the KD and BU measures (both individually and combined) by large margins on all attack strategies tested, across all datasets tested. For the most effective attack strategy known to date, the Opt attack, the LID-based detector achieved AUC scores of 99.24%, 98.94% and 97.60% on MNIST, CIFAR-10 and SVHN respectively, compared to AUC scores of 95.35%, 93.77% and 90.66% for the detector based on KD and BU. And the performance of LID-based detector over five random runs is table. This strong performance suggests that LID is a highly promising characteristic for the discrimination of adversarial examples and regions. We also note that KD was not effective for
3.4. EVALUATING LID-BASED CHARACTERIZATION OF ADVERSARIAL EXAMPLES

Table 3.2: A comparison of the discrimination power (AUC score (%) of a logistic regression classifier) among LID, KD, BU, and KD+BU. The average AUC score with standard deviation (±std) over 5 random runs is computed for each attack strategy on each dataset, and the best results are highlighted in bold.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Feature</th>
<th>FGM</th>
<th>BIM-a</th>
<th>BIM-b</th>
<th>JSMA</th>
<th>Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>KD</td>
<td>78.12 ± 1.2</td>
<td>98.14 ± 0.7</td>
<td>98.61 ± 0.2</td>
<td>68.77 ± 2.3</td>
<td>95.15 ± 1.3</td>
</tr>
<tr>
<td></td>
<td>BU</td>
<td>32.37 ± 0.9</td>
<td>91.55 ± 0.1</td>
<td>25.46 ± 1.2</td>
<td>88.74 ± 1.2</td>
<td>71.30 ± 0.7</td>
</tr>
<tr>
<td></td>
<td>KD+BU</td>
<td>82.43 ± 1.5</td>
<td>99.20 ± 0.1</td>
<td>98.81 ± 0.8</td>
<td>90.12 ± 2.5</td>
<td>95.35 ± 1.2</td>
</tr>
<tr>
<td></td>
<td>LID</td>
<td>96.89 ± 1.2</td>
<td>99.60 ± 0.4</td>
<td>99.83 ± 0.6</td>
<td>92.24 ± 1.4</td>
<td>99.24 ± 0.6</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>KD</td>
<td>64.92 ± 1.7</td>
<td>68.38 ± 2.0</td>
<td>98.70 ± 0.9</td>
<td>85.77 ± 3.7</td>
<td>91.35 ± 1.2</td>
</tr>
<tr>
<td></td>
<td>BU</td>
<td>70.53 ± 2.7</td>
<td>81.60 ± 1.3</td>
<td>97.32 ± 1.0</td>
<td>87.36 ± 3.2</td>
<td>91.39 ± 2.5</td>
</tr>
<tr>
<td></td>
<td>KD+BU</td>
<td>70.40 ± 1.2</td>
<td>81.33 ± 2.1</td>
<td>98.90 ± 0.4</td>
<td>88.91 ± 1.6</td>
<td>93.77 ± 2.9</td>
</tr>
<tr>
<td></td>
<td>LID</td>
<td>82.38 ± 1.5</td>
<td>82.51 ± 2.4</td>
<td>99.78 ± 0.4</td>
<td>95.87 ± 0.3</td>
<td>98.94 ± 1.2</td>
</tr>
<tr>
<td>SVHN</td>
<td>KD</td>
<td>70.39 ± 4.2</td>
<td>77.18 ± 3.7</td>
<td>99.57 ± 0.9</td>
<td>86.46 ± 2.3</td>
<td>87.41 ± 1.9</td>
</tr>
<tr>
<td></td>
<td>BU</td>
<td>86.78 ± 2.2</td>
<td>84.07 ± 0.9</td>
<td>86.93 ± 1.4</td>
<td>91.33 ± 2.2</td>
<td>87.13 ± 2.8</td>
</tr>
<tr>
<td></td>
<td>KD+BU</td>
<td>86.86 ± 1.7</td>
<td>83.63 ± 2.6</td>
<td>99.52 ± 0.7</td>
<td>93.19 ± 0.9</td>
<td>90.66 ± 1.4</td>
</tr>
<tr>
<td></td>
<td>LID</td>
<td>97.61 ± 0.8</td>
<td>87.55 ± 2.0</td>
<td>99.72 ± 0.1</td>
<td>95.07 ± 1.1</td>
<td>97.60 ± 0.9</td>
</tr>
</tbody>
</table>

the FGM, JSMA and BIM-a attack strategies, whereas the BU measure failed to detect most FGM and BIM-b attacks on the MNIST dataset.

3.4.3.2 Generalizability analysis

It is natural to consider the question of whether samples of one attack strategy may be detected by a model that has been trained on samples of a different attack strategy. We conduct a preliminary investigation of this issue by studying the generalizability of KD, BU and LID for detecting previously unseen attack strategies on the CIFAR-10 dataset. The KD, BU and LID detectors are trained on samples of the simplest attack strategy, FGM, and then tested on samples of the more complex attacks BIM-a, BIM-b, JSMA and Opt. The training and test datasets are generated in the same way as in our previous experiments with only the FGM attack applied on the train set while the other attacks applied separately on the test set. The test attack data is standardized by scaling so as to fit the training data. The results are shown in Table 3.3 from which we see that the LID detector trained on FGM can accurately detect the much more complex
3. CHARACTERIZING ADVERSARIAL SUBSPACES USING LOCAL INTRINSIC DIMENSIONALITY

Table 3.3: This table of AUC scores (%) shows the generalizability of detectors trained on the FGM attack strategy (row) to other forms of attack (column), with respect to the CIFAR-10 dataset. The mean and standard deviation (±std) over five random runs are reported, and the best results are indicated in bold font.

<table>
<thead>
<tr>
<th>Train</th>
<th>Test</th>
<th>FGM</th>
<th>BIM-a</th>
<th>BIM-b</th>
<th>JSMA</th>
<th>Opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGM</td>
<td>KD</td>
<td>64.92 ± 4.9</td>
<td>69.15 ± 5.1</td>
<td>89.71 ± 4.9</td>
<td>85.72 ± 5.4</td>
<td>91.22 ± 4.2</td>
</tr>
<tr>
<td></td>
<td>BU</td>
<td>70.53 ± 6.1</td>
<td>81.67 ± 5.3</td>
<td>2.65 ± 1.5</td>
<td>86.79 ± 3.6</td>
<td>91.27 ± 2.3</td>
</tr>
<tr>
<td></td>
<td>LID</td>
<td>82.38 ± 3.4</td>
<td>82.30 ± 2.9</td>
<td>91.61 ± 2.0</td>
<td>89.93 ± 2.3</td>
<td>93.32 ± 2.6</td>
</tr>
</tbody>
</table>

attacks of the other strategies. The KD and BU characteristics can also achieve good performance on this transfer learning task, but are less consistent than our proposed LID characteristic. The results appear to indicate that the adversarial regions generated by different attack strategies possess similar dimensional properties.

It is worth mentioning that the BU detector trained on the FGM attack generalizes poorly to detect BIM-b adversarial examples (AUC=2.65%). This may due to the fact that BIM-b performs a fixed number of perturbations (50 in our setting) that likely extend well beyond the classification boundary. Such perturbed adversarial examples tend to possess Bayesian model uncertainties even lower than normal examples under dropout randomization, as dropping out a certain proportion of their representations (50% in our setting) would not lead to high prediction variance. This is consistent with the results reported in [47]: only 4% of BIM-b adversarial examples, in contrast to at least 74.7% of adversarial examples of other attack strategies, exhibit higher Bayesian uncertainties than normal examples. It is particularly interesting to see that detectors trained on the FGM attack strategy can sometimes achieve better performance when used to identify the other attacks. An extensive study of detection generalizability across all attack strategies is an interesting topic for future work.

3.4.3.3 Effect of larger minibatch sizes in LID estimation

In the estimation of LID values, a default minibatch size of 100 was used, with a view to ensuring efficiency. Even though experimental analysis has shown that the MLE estimator of LID is not stable on such small samples [3], this is more
3.4. EVALUATING LID-BASED CHARACTERIZATION OF ADVERSARIAL EXAMPLES

Figure 3.5: The detection AUC score of LID estimated using different neighborhood sizes $k$ with a larger minibatch size of 1000. The results are shown for the detection of Opt attacks on the MNIST, CIFAR-10 and SVHN datasets.

than adequately compensated for by the learning process in LID-based detection, as evidenced by the superior performance shown in Table 3.2. However, it is an interesting question as to whether the use of larger minibatch sizes could further improve the performance (as measured by AUC) without incurring unreasonably high computational cost. Figure 3.5 shows the discrimination power (detection AUC) of LID characteristics estimated using two different minibatch sizes: the default setting of 100, and a larger size of 1000. The horizontal axis represents different choices of the neighborhood size $k$, from 10% to 90% percent to the batch size. It does indicate that increasing the batch size can improve the detection performance even further (the peak AUC is higher for the larger minibatch size). A comprehensive investigation of the tradeoffs among minibatch size, LID estimation accuracy, and detection performance is an interesting direction for future work.

3.4.3.4 Adaptive attack against LID measurement

To further evaluate the robustness of our LID-based detector, we applied an adaptive Opt attack in a white-box setting. Similar to the strategy used in [21] to attack the KD-based detector, we used an Opt $L_2$ attack with a modified adversarial objective:

$$\minimize_{x} \|x - x_{adv}\|_2^2 + \alpha \cdot (\ell(x_{adv}) + \text{LID}(x_{adv}))$$ (3.5)
Table 3.4: The attack failure rate (AFR) in percentage (%) of an adaptive attack targeting the LID-based detector. The mean and standard deviation ($\pm$std) over five random runs were reported.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>MNIST</th>
<th>CIFAR-10</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (LID at all layers): AFR</td>
<td>99.44 ± 0.22</td>
<td>99.48 ± 0.25</td>
<td>99.59 ± 0.29</td>
</tr>
<tr>
<td>2 (LID at one layer): AFR</td>
<td>98.89 ± 2.99</td>
<td>95.7 ± 2.10</td>
<td>97.4 ± 1.56</td>
</tr>
</tbody>
</table>

where $\ell(x_{adv})$ is the original adversarial objective of Opt attack, $\alpha$ is a constant balancing between the amount of perturbation and the adversarial strength, and the LID scores are computed at the pre-softmax layer.

We test two different scenarios for detection. In the first scenario, we use LID features as described in Algorithm 3.1. In the second scenario, we use LID scores only at the pre-softmax layer. Since the Opt attack uses only the pre-softmax activation output to guide the perturbation, the latter scenario allows a fair comparison to be made [22, 21]. The optimal constant $\alpha$ is determined via an internal binary search for $\alpha \in [10^{-3}, 10^{6}]$. The rationale for the minimization of the LID characteristic in Equation (3.5) is that adversarial examples have higher LID characteristics than normal examples, as we have demonstrated in Section 3.4.2.

We applied the adaptive attack on 1000 normal images randomly chosen from the detection test set ($test$). The deep networks used were the same ConvNet configurations as used in our previous experiments. To evaluate attack performance, instead of AUC as measured in the previous sections, we report accuracy as suggested by [21]. We see from Table 3.4 that the adaptive attack in Scenario 2 fails to find any valid adversarial example 100%, 95.7% and 97.2% of the time on MNIST, CIFAR-10 and SVHN respectively. In addition, when trained on all transformation layers (Scenario 1), the LID-based detector still correctly detected the attacks 100% of the time. Based on these results, we can conclude that integrating LID into the adversarial objective (increasing the complexity of the attack) does not make detection more difficult for our method. This is in contrast to the work of [21], who showed that incorporating kernel density into the objective function makes detection substantially more difficult for the KD method.
3.5 Chapter summary

In this chapter, we have studied the vulnerability of deep networks to adversarial perturbations. We addressed the challenge of understanding the properties of adversarial regions, particularly with a view to detecting adversarial examples. We characterized the dimensional properties of adversarial regions via the use of Local Intrinsic Dimensionality (LID), and showed how these could be used as features in an adversarial example detection process. Our empirical results suggest that LID is a highly promising measure for the characterization of adversarial examples, one that can be used to deliver state-of-the-art discrimination performance. From a theoretical perspective, we have provided an initial intuition as to how LID is an effective method for characterizing adversarial attack, one which complements the recent theoretical analysis showing how increases in LID effectively diminish the amount of perturbation required to move a normal example into an adversarial region (with respect to 1-NN classification) [2]. Further investigation in this direction may lead to new techniques for both adversarial attack and defense.
3. CHARACTERIZING ADVERSARIAL SUBSPACES USING LOCAL INTRINSIC DIMENSIONALITY
Chapter 4

Adversarial Feedback Generation with Neural Networks

In this chapter, we explore the novel use of adversarial techniques in a beneficial way to generate feedback for intelligent tutoring in simulation-based learning (or SBL) environments, in contrast to crafting adversarial examples to fool neural networks. As we discussed in Section 2.3.2 for an SBL platform to be fully utilized as an effective training tool, it is essential that feedback on performance is provided automatically in real-time during training. It is the aim of this work to develop an efficient and effective feedback generation method for the provision of real-time feedback in SBL. Existing methods either have low effectiveness in improving novice skills or suffer from low efficiency, resulting in their inability to be used in real-time (see Table 2.1 for a summary).

In this chapter, we propose a neural network based method to generate feedback using the adversarial technique. The proposed method utilizes a bounded adversarial update to minimize a $L_1$ regularized loss via back-propagation. We empirically show that the proposed method can be used to generate simple, yet effective feedback. Also, it was observed to have high effectiveness and efficiency when compared to existing methods, thus making it a promising option for real-time feedback generation in SBL.\(^1\)

\(^1\)This chapter of thesis is based on the following published paper: X. Ma, S. Wijewickrema, S. Zhou, Y. Zhou, Z. Mhammedi, S. O’Leary, J. Bailey, “Adversarial generation of real-time feedback with neural networks for simulation-based training,” in Pro-
4.1 Introduction

As we introduced in Section 2.3.3, feedback generation in SBL has three challenges. First, feedback should be generated in a timely manner as delayed feedback can lead to confusion or even cause fatal consequences in reality. An acceptable time-limit is 1 second after inappropriate action is detected. This is because feedback should be provided before the learner makes the next move [166]. Second, feedback should be actionable instructions that can be followed by the trainee to improve skills or correct mistakes. This is because SBL tasks often consist of a series of delicate operations that require precise instructions. Third, feedback should be simple, referring to only a few aspects of the skill, as practically people cannot focus on many things at a time. Also, this reduces distractions to the trainee and decreases cognitive load, thus increasing the usefulness of the feedback [199].

In this chapter, we propose a neural network based method to generate feedback using the adversarial technique. As discussed in Section 2.2, one intriguing property of neural networks is that the input can be changed by maximizing the prediction error so that it moves into a different class with high confidence [202] (see Section 2.2 for detailed discussion). This property has been used to generate adversarial examples from deep neural nets in image classification [63]. An adversarial example is formed by applying small perturbations (imperceptible to the human eye) to the original image, such that the neural network misclassifies it with high confidence. Although the adversarial example has similarities to the feedback problem in that they both change the input to a different class, they are not synonymous. First, the adversarial example is formed by adding intentionally-designed noise that may result in states that do not exist or have practical meaning in a real-world dataset such as that of the feedback problem. Second, only a few changes to inputs are recommended for feedback, to make it useful to follow.
4.2 Feedback generation problem

Recall that, Figure 2.11 demonstrates a typical real-time feedback process in SBL, and this chapter focuses on the “feedback generation method” part of the system as highlighted in the figure. Here, we follow the problem formulation in Problem 2.3.1, where the feedback generation problem was defined from an ‘expert-novice’ perspective. We acknowledge that there may be more than 2 levels of expertise in some SBL applications. However, this can be easily addressed using the one-vs-rest approach.

In SBL, the feedback generation problem is to find the optimal action that can be taken to change a novice instance to an expert instance. Suppose the dataset \( X \in \mathbb{R}^d \) consists of \( d \) features, \( N \) instances defined by the feature vector \( x = (x_1, ..., x_d) \), and associated with a class label \( y \in \{0, 1\} \) (1: expert, 0: novice). \( F(x) \) is a prediction model learned over \( X \). The feedback generation problem can then be defined as in Problem 2.3.1. In SBL, \( \ell(x, x') = \|x - x'\|_0 \), i.e., the number of changed features. The cost limit \( C \) is often a small integer such as 1 or 2 in SBL so as to meet the requirements discussed in Section 4.1.

4.3 Proposed method

To tackle the feedback generation problem, we propose the use of a neural network as the prediction model \( H(x) \) and introduce a method that directly generates feedback from the neural network. Let \( H_\theta(x) \), with parameters/weights \( \theta \), be the neural network learned with respect to the loss function \( J(\theta, x, y) \), where \( x \) is the input or feature vector, \( y \) the class value associated with \( x \), and \( y^* \) the target class we want \( x \) to be in.

Recall that during the training process, the weights \( \theta \) are updated so that the loss \( J(\theta, x, y) \) is minimized. Therefore, if we keep \( \theta \) fixed while the input \( x \) is updated so that \( J(\theta, x, y) \) is maximized, we can get a new instance that has high confidence of being in the opposite class to its original class \( y \). To maximize \( J(\theta, x, y) \), the input can be updated in the positive direction of the gradient following Equation (4.1), where \( \epsilon \) is the learning rate.
This is the property that has been used to generate adversarial examples in image classification. Since adversarial examples require small perturbations in the input image, [63] applied a sign function to linearize the loss function around the current value of $\theta$, as shown in Equation (4.2). This method updates all the pixels of the input image once to get small perturbations.

$$x = x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$$  \hspace{1cm} (4.2)

Equation (4.1) works well for two-class tasks. However, for multi-class tasks, there are more than one opposite classes to $y$. This means using Equation (4.1) cannot guarantee the new instance has high confidence in the target class $y^\ast$. The alternative is minimizing the loss $J(\theta, x, y^\ast)$ with respect to the specific target class $y^\ast$ as defined in Equation (4.3).

$$x = x - \epsilon \nabla_x J(\theta, x, y^\ast)$$  \hspace{1cm} (4.3)

Although Equation (4.3) works for both two-class and multi-class tasks, it still has two potential problems that limit its use for feedback generation in SBL. First, it may change all input features, thus violating the constraint (e.g., $\|x - x'\|_0 \leq C$) of Problem 2.3.1. Second, the update may explode the values of inputs to extremely small or large values, similar to the exploding gradient problem [156]. However, in practice, some features may have a certain value range outside of which the feature is meaningless.

To solve the first problem, we introduce a $L1$ regularization term to $J(\theta, x, y^\ast)$ to control the sparsity of the change so as to generate simple feedback. The new loss function is defined in Equation (4.4), where $\lambda$ is the regularization parameter and $x_0$ is the original input that needs to be changed.

$$J'(\theta, x, y^\ast) = J(\theta, x, y^\ast) + \lambda \|x - x_0\|_1$$  \hspace{1cm} (4.4)

To solve the second problem, we propose a bounded update approach (see Equations (4.5) and (4.6)) as an alternative to Equation (4.3). It incorporates the
value range (defined by lower and upper bounds) of a feature into the update to ensure the updated feature value is still within range.

\[ x = x - \epsilon S_x \left( x S_x - \frac{a}{2} (1 + S_x) + \frac{b}{2} (1 - S_x) \right) \] (4.5)

\[ S_x = \text{sign}(\nabla_x J'(\theta, x, y^*)) \] (4.6)

\( S_x \) is the sign of the partial derivative of \( J'(\theta, x, y^*) \) with respect to \( x \). The upper and lower bounds of \( x \) are \( a \) and \( b \) respectively, i.e. \( x_i \in [a_i, b_i] \).

According to Equation (4.5), if the gradient \( \nabla_x J'(\theta, x, y^*) \) is positive (i.e., \( S_{x_i} = 1 \)), the update will become \( x_i = x_i - \epsilon (x_i - a_i) \) which means \( x_i \) moves a small step towards its lower bound \( a_i \). Similarly, a negative gradient gives \( x_i = x_i + \epsilon (b_i - x_i) \), a move towards its upper bound \( b \). No update will be applied if the gradient is zero, as in this case, \( S_{x_i} = 0 \). This bounded update not only guarantees the correct update direction to minimize the loss, but also ensures that \( x_i \in [a_i, b_i] \) always holds true (see Lemma 1 and proof).

**Lemma 1.** If \( a_i \leq x_i \leq b_i \), \( 0 < \epsilon \ll 1 \), and \( x'_i = x_i - \epsilon S_{x_i} (x_i S_{x_i} - \frac{a_i}{2} (1 + S_{x_i}) + \frac{b_i}{2} (1 - S_{x_i})) \), then \( a_i \leq x'_i \leq b_i \).

**Proof.** The sign function \( S_{x_i} \) only has 3 outputs: 1,0 or -1.

**Case 1.** If \( S_{x_i} = 0 \), then \( x'_i = x_i \).

In this case, \( a_i \leq x'_i = x_i \leq b_i \) holds true.

**Case 2.** If \( S_{x_i} = 1 \), then \( x'_i = x_i - \epsilon (x_i - a_i) \).

In this case, \( x'_i - a_i = (1 - \epsilon) (x_i - a_i) \) and \( x'_i - b_i = \epsilon (a_i - x_i) + (x_i - b_i) \). Then, \( a_i \leq x_i \leq b_i \) and \( 0 < \epsilon \ll 1 \) gives \( x_i - a_i \geq 0 \), \( x_i - b_i \leq 0 \) and \( 1 - \epsilon > 0 \). Therefore, \( x'_i - a_i \geq 0 \) and \( x'_i - b_i \leq 0 \), that is, \( a_i \leq x'_i \leq b_i \).

**Case 3.** If \( S_{x_i} = -1 \), then \( x'_i = x_i + \epsilon (b_i - x_i) \).

And in this case, \( x'_i - a_i = \epsilon (b_i - x_i) + (x_i - a_i) \) and \( x'_i - b_i = (1 - \epsilon) (x_i - b_i) \). Similarly, \( b_i - x_i \geq 0 \), \( x_i - a_i \geq 0 \) and \( 1 - \epsilon > 0 \) gives \( x'_i - a_i \geq 0 \) and \( x'_i - b_i \leq 0 \), that is, \( a_i \leq x'_i \leq b_i \).
To conclude, in all cases, \( a_i \leq x'_i \leq b_i \) holds true.

Equations (4.4), (4.5) and (4.6) give the definition of the proposed “neural network-based feedback (NNFB) method”. NNFB takes a novice instance \( x \) as input, iteratively updates \( x \) (different from the one-time-update in generating adversarial examples) until it converges or meets the terminating criteria. Let the generated new instance be \( x' \), the feedback is then the action \( A : x \rightarrow x' \) (see the example in Problem 2.3.1).

When feedback is delivered, we need to ensure that it contains only \( C \) features. Although, the \( L1 \) regularization reduces the number of feature changes in general, in the absence of valid feedback with a low number of feature changes, it may still result in ones with higher numbers of feature changes. To overcome this issue, we suggest a post-selection process that iteratively tests all feature changes and select the ones with \( C \) or less changes that result in the best improvements.

The proposed method (NNFB) is easily generalizable to different SBL applications. First, the regularization term in \( J'(\theta, x, y^*) \) can be adjusted accordingly for different applications (for example, \( L2 \) norm for applications that prefer small changes). Furthermore, NNFB offers flexible control over feature changes as the lower and upper bounds are adjustable for different features and even for different input instances. For example, we can set \( a_i = b_i = x_i \) for a categorical feature that cannot be changed, such as prior simulation experience. This flexibility also benefits those applications that have discrete cost functions as some explicit cost limits can be easily incorporated into the bounds.

4.4 Experimental validation

In this section, we first describe the two real-world datasets that were used in the experiments. Then, we briefly introduce the existing methods that the proposed method was compared against, followed by the experimental setup. Finally, we discuss the experiment results.
4.4. EXPERIMENTAL VALIDATION

Figure 4.1: The University of Melbourne Virtual Reality Temporal Bone Surgery Simulator: it consists of a computer that runs a 3D model of a human temporal (ear) bone and a haptic device that provides tactile resistance to simulate drilling.

4.4.1 Datasets

We tested our method on two real-world SBL datasets. These datasets were collected from a temporal bone surgical simulator designed to train surgeons in ear-related surgeries, as illustrated in Figure 4.1 (more details of the simulator can be found in Section 2.3.2 and Figure 2.10). 7 expert and 12 novice surgeons performed two different surgeries that require very different surgical skills: cortical mastoidectomy - dataset 1 ($D_1$) and posterior tympanotomy - dataset 2 ($D_2$). Surgical skill is defined by 6 numeric skill metrics: stroke length, drill speed, acceleration, time elapsed, the straightness of the trajectory and drill force. The skill metrics were recorded by the simulator at a rate of approximately 15 Hz. Overall, $D_1$ includes 60K skill instances (28K expert and 32K novice) while $D_2$ includes 14K skill instances (9K expert and 5K novice). Both datasets were normalized to the range of [0, 1] using feature scaling as follows.

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

(4.7)

where, $x$ and $x'$ are the original and scaled feature vectors respectively and $x_{\text{min}}$
and $x_{\text{max}}$ are the minimum and maximum feature values of $x$ respectively.

### 4.4.2 Compared methods

Existing feedback generation methods compared with NNFB are as follows.

- **Split Voting (SV):** This is the random forest based state-of-the-art generation method for providing real-time feedback \[236\] as discussed in Section 2.3.3.

- **Integer Linear Programming (ILP):** This method solves the random forest feedback generation problem by transforming it to an integer linear programming problem \[31\] as discussed in Section 2.3.3.

- **Random Iterative (RI):** This method randomly selects a feature and iteratively selects the best value among the feature’s value partitions in the random forest \[31\].

- **Random Random (RR):** This method randomly picks a feature from a novice instance and selects a random change to that feature as the suggested feedback.

### 4.4.3 Experimental setup

For testing, we randomly chose one novice participant, then took all instances performed by this novice as the test set. The remaining instances were used for training. This simulates the real-world scenario of an unknown novice using the simulator. Parameter tuning was performed on the training data based on a 11-fold leave-one-novice-out cross-validation. In each fold, we took all instances from one randomly chosen novice as the validation set.

All methods were restricted to generate feedback with only one feature change, which is a typical requirement in SBL. This is a binary task as there are only 2 skill levels (expert and novice). All methods were then evaluated using 2 measures: 1) efficiency and 2) effectiveness. Overall, a good feedback generation method should have high effectiveness and high efficiency (low time-cost). Note that effective generation of feedback not only means the generation
method itself is effective but also indicates that the base classifier trained to discriminate novices from experts is accurate.

Efficiency was measured using the time-cost (in seconds) spent on average to generate feedback for one novice instance. The novice instance $x$ will be changed to the target instance $x'$ by the feedback $A : x \rightarrow x'$ (see Problem 2.3.1). Thus, we use the quality of the target instances (i.e., $\{x'\}$) to measure the effectiveness of the feedback. As defined in Equation (4.8), effectiveness $\mathcal{E}$ is the percentage of expert instances in $\{x'\}$.

$$\mathcal{E} = \frac{|\{x' | x' \text{ is an expert instance}\}|}{|\{x'\}|} \quad (4.8)$$

However, how instances are classified is dependent on the classifier used. To obtain more convincing results, we used 6 classifiers of different types for evaluation. The evaluation classifiers are: neural network (NN), random forest (RF), logistic regression (LR), SVM (RBF kernel), naïve Bayes (NB) and KNN ($K = 10$). A generation method that scores consistently high levels of $\mathcal{E}$ across classifiers is deemed effective.

Experiments were carried out on a typical PC with 2.40GHz CPU. The ILP solver used for the ILP method was CPLEX\footnote{https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer} as suggested by the authors, and the neural network/random forest implementations we used were from scikit-learn. Default settings in scikit-learn were used for parameters not specifically mentioned here.

### 4.4.4 Parameter tuning

Parameter tuning was performed on the training data with a 11-fold leave-one-novice-out cross-validation as mentioned above in Section 4.4.3. A two-layer neural network architecture was used for NNFB. For $\mathcal{D}_1$, a neural network with 250 hidden neurons was selected for NNFB while a random forest with 120 trees was selected for SV, RS and ILP. For $\mathcal{D}_2$, NNFB used a neural network with 120 hidden neurons while SV, RS and ILP used a random forest with 100 trees. These parameters were selected based on the turning point of the number of hidden
neurons or the number of trees with respect to the mean squared error (MSE) of the neural network and random forest respectively.

In terms of the regularization parameter $\lambda$ in NNFB, Figure 4.2 indicates that larger $\lambda$ results in simple feedback with a fewer number of feature changes. When $\lambda = 1$, a feedback on average consists of only one feature change, but remains highly confident ($F(x') > 0.7$) to change a novice instance to an expert instance. Therefore, we chose $\lambda = 1$ for NNFB. Since datasets have been normalized, the upper bounds for all features are 1 and the lower bounds are 0. Other settings for NNFB include Rectified Linear Unit (ReLU) activation function [59], cross entropy loss and learning rate $\epsilon = 1 \times 10^{-4}$.

### 4.4.5 Results

We first demonstrate the overall performance considering both effectiveness and efficiency. Figure 4.3 illustrates the effectiveness of each method as evaluated using 6 different evaluation classifiers with respect to the time-cost (inverses to efficiency) for each dataset. As seen in the figure, the proposed method shows the desired performance of highest effectiveness at an acceptably low
Table 4.1: The effectiveness (mean±std) tested by 6 evaluation classifiers on dataset $\mathcal{D}1$. The best results are highlighted in bold.

<table>
<thead>
<tr>
<th>Test Classifier</th>
<th>$\mathcal{D}1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR</td>
</tr>
<tr>
<td>NN</td>
<td>0.19±0.06</td>
</tr>
<tr>
<td>RF</td>
<td>0.23±0.10</td>
</tr>
<tr>
<td>LR</td>
<td>0.35±0.07</td>
</tr>
<tr>
<td>SVM</td>
<td>0.27±0.06</td>
</tr>
<tr>
<td>NB</td>
<td>0.32±0.12</td>
</tr>
<tr>
<td>KNN</td>
<td>0.30±0.05</td>
</tr>
</tbody>
</table>

Table 4.2: The effectiveness (mean±std) tested by 6 evaluation classifiers on dataset $\mathcal{D}2$. The best results are highlighted in bold.

<table>
<thead>
<tr>
<th>Test Classifier</th>
<th>$\mathcal{D}2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR</td>
</tr>
<tr>
<td>NN</td>
<td>0.98±0.04</td>
</tr>
<tr>
<td>RF</td>
<td>1.00±0.00</td>
</tr>
<tr>
<td>LR</td>
<td>0.87±0.08</td>
</tr>
<tr>
<td>SVM</td>
<td>0.96±0.04</td>
</tr>
<tr>
<td>NB</td>
<td>0.76±0.12</td>
</tr>
<tr>
<td>KNN</td>
<td>0.76±0.08</td>
</tr>
</tbody>
</table>

time-cost (within the real-time time-limit) when compared to the other methods. This proves that the adversarial technique can be used to generate effective and timely feedback for SBL. Note that the slightly higher variance of the NNFB method indicates the varying resistance of test classifiers to adversarial generation.

Detailed results for effectiveness of the feedback generation methods across the 6 evaluation classifiers is shown in Table 4.1 and 4.2 for the datasets $\mathcal{D}1$ and $\mathcal{D}2$. On both datasets, NNFB achieved comparable performance to ILP and outperformed all others methods across all classifiers. However, as shown in Table 4.3, ILP violates the real-time time-limit as discussed in Section 2.3.1 and as such, will not be suitable for most SBL applications. Although both RR
4. ADVERSARIAL FEEDBACK GENERATION WITH NEURAL NETWORKS

Figure 4.3: Box plot representing the performance of the 6 evaluation classifiers with respect to effectiveness and time-cost. Each method has two boxes that represent the 2 datasets $D_1$ and $D_2$. Colored view is recommended.

Table 4.3: The time-cost (mean±std in seconds) on average for generating one feedback, tested on datasets $D_1$ and $D_2$.

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>0.013±0.004</td>
<td>0.014±0.001</td>
</tr>
<tr>
<td>RI</td>
<td>0.504±0.098</td>
<td>0.401±0.020</td>
</tr>
<tr>
<td>SV</td>
<td>0.023±0.003</td>
<td>0.017±0.003</td>
</tr>
<tr>
<td>ILP</td>
<td>31.738±2.439</td>
<td>27.760±3.107</td>
</tr>
<tr>
<td>NNFB</td>
<td>0.142±0.029</td>
<td>0.121±0.016</td>
</tr>
</tbody>
</table>

and SV are more efficient than the proposed method in feedback generation, they show significantly lower levels of effectiveness when compared to NNFB. Thus, it can be concluded that in terms of both effectiveness and efficiency, the proposed method is the best suited for providing real-time feedback in SBL applications.
4.5 Chapter summary

In this chapter, we have explored the possibility of using adversarial perturbation techniques in a beneficial way to generate tutoring feedback, in contrast to its use for attacking machine learning models. We introduced a technique for the adversarial generation of real-time feedback with neural networks for SBL. The proposed method (NNFB) applies a bounded adversarial update on the novice skill vector to generate an optimal expert skill vector in order to be used in the provision of feedback. To ensure that the suggested action is simple enough to practically undertake, we adopted $L_1$ regularization to obtain feedback with a fewer number of feature changes. We explored theoretically the validity of NNFB, and showed empirically that it outperforms existing methods in providing effective real-time feedback.
Chapter 5

Adversarial Feedback Generation with Random Forests

In this chapter, we further explore the possibility of using adversarial techniques with traditional machine learning models, i.e., RFs, for feedback generation. As we discussed in Section 2.3.1, RF models do not have closed-form functions and computable gradients that can be exploited by adversarial techniques. In this work, we demonstrate that feedback perturbations can still be extracted from RFs efficiently, based on a geometric exploration of RF decision space.

Here, the problem we aim to solve is still the standard feedback generation problem as in Problem 2.3.1, the application context is still SBL (see the virtual reality based surgical platform in Figure 2.10 for an example). Instead of working with standard adversarial techniques with gradients, here we need to develop a perturbation method that can change an instance from ‘novice’ class to ‘expert’ class in RF model, but does not rely on gradients. [1]

5. ADVERSARIAL FEEDBACK GENERATION WITH RANDOM FORESTS

5.1 Introduction

As introduced in Section 2.3.3, SBL have become a powerful platform for training [164, 221, 126]. In a VR-based simulation, trainees can practice on multiple surgical cases in a risk-free, immersive, and interactive learning environment. However, it still requires the supervision of human experts to ensure that feedback is provided during training to facilitate the acquisition of proper surgical skills. To reduce the reliance on the availability of human experts, the provision of feedback should be automated.

One of the simplest ways of providing real-time feedback in VR simulation is the “follow-me” approach [164]. It provides a ‘ghost’ drill recorded by an expert to lead the trainee through the surgery. However, trainees who are unfamiliar with the procedure may struggle to follow the pace of the ghost expert. Other works utilized data mining techniques to generate feedback that can change adaptively in response to trainee’s performance. For example, a pattern mining algorithm was applied to discover expert and novice skill patterns to support feedback provision in a surgical simulation [237]. However, experts and novices often share a considerable amount of similar behaviour which makes it difficult to identify significant patterns. This effect can be reduced through the use of random forest (RF) based methods such as Split Voting (SV) [236]. SV first trains a RF prediction model that can distinguish expert skill from novice skill, then uses the prediction model to formulate feedback. Although this method is efficient, the effectiveness of formulated feedback in improving novice skill is low. In fact, extracting feedback from a RF is a NP-hard problem. However, it can be solved by high-performance ILP solvers when transformed to an integer linear programming (ILP) problem [31]. This approach is highly effective in improving novice skills as it searches for the global optimal solution but the searching process is computationally expensive. Our previous work in Chapter 4 shows that neural networks can also be used to generate feedback for simulation-based training.

As discussed in Section 2.3.3, there are three challenges in providing real-time feedback in a simulation environment: 1) feedback should be effective so it can improve novice skill to expert skill, 2) feedback should be provided in
real-time (within 1 second) when novice skill is detected, as delayed feedback may result in confusion or cause undesirable consequences [236], and 3) feedback should be simple (ideally based on one aspect of performance) so that the cognitive load is manageable [199].

The ideal feedback generation method would be highly effective, yet efficient enough to be used in real-time. Highly effective methods exist but they typically lack sufficient efficiency and vice versa. Thus, the key in real-time feedback generation is to find the right balance between the two, which to our knowledge has not been adequately addressed in the literature. To overcome this, we propose a geometric method based on decision space discretization to formulate feedback using a RF, and demonstrate that it has near-optimal effectiveness, is highly efficient, and scalable.

The simulation used here is a Temporal Bone Surgery (TBS) Simulator as illustrated in Figure 2.10 and discussed in Section 2.3.2. It consists of a 3D temporal (ear) bone model running on a computer and a haptic device providing tactile resistance to simulate drilling. The surgery we focus on is cortical mastoidectomy, which involves drilling parts of the temporal bone. In TBS, surgical skill is defined using characteristics of a stroke (e.g. length, speed, acceleration, duration, straightness, force). A stroke is a continuous drilling motion in the same general direction that results in material removal [238]. Feedback is the action that can improve a novice stroke to an expert stroke.

An overview of the feedback generation process is shown in Figure 2.11. The generation method is trained offline over labelled (expert/novice) strokes. This is used in real-time to provide feedback that improves novice behaviour. The feedback generation problem is to find the best change in stroke characteristics that changes a novice stroke to an expert stroke, as defined in Problem 2.3.1.

\[ \ell(x, x') = \|x - x'\|_0 \leq C = 1 \]

is the cost constraint that only allows a change in a single feature, to minimise cognitive load [199].

### 5.2 Random forest based feedback generation

We propose the use of RF as the prediction model \( F(x) \). In contrast to existing methods [236, 31, 229], we analyze the RF-based feedback generation prob-
lem from a geometric point of view and introduce an approximation method to solve it.

5.2.1 A geometric view

A RF prediction model is an ensemble of decision trees with each tree trained over a random subset of labelled strokes and features (see Figure 5.1). The leaf node of the tree can be seen as a hyper-rectangle defined by the decision path from the root to the node. As in the example, the expert node (green rectangle with label 1) in Tree 1 can be represented by rectangle \( \{ x_1 > 0.5, x_2 \leq 0.2 \} \). Thus, a RF can be seen as a union of hyper-rectangles overlapping in the data space \([16]\). Similar to a leaf node, a hyper-rectangle has an associated class label indicating the expertise level of strokes within it. Let \( \mathcal{R}^1 / \mathcal{R}^0 \) denote the hyper-rectangle with a expert/novice class label respectively.

A RF divides the data space into expert and novice subspaces. Expert subspaces are small areas that are overlapped by more \( \mathcal{R}^1 \)'s than \( \mathcal{R}^0 \)'s. Thus, the most expert-like strokes can be found in the areas that are overlapped by the most \( \mathcal{R}^1 \)'s. Then, feedback can be interpreted as moving a stroke from novice subspace to areas that are overlapped by the most \( \mathcal{R}^1 \)'s. However, calculating all
possible intersections between $R^1$s is NP-hard. To overcome this, we propose an approximation method as follows that uniformly takes a few points from $R^1$ to represent them and finds a solution based on those representatives.

### 5.2.2 Random forest discretization

This step discretizes the expert hyper-rectangles of a RF to a finite number of integer-represented points. This transformation allows the uniform selection of a few points from an expert hyper-rectangle as its representatives. In a RF, a feature value is automatically segmented into multiple partitions by the split nodes. Suppose the number of partitions of the $d$ features are $m_1, ..., m_d$ respectively. We define an integer variable $p_i \in [1, m_i]$ for feature $x_i$ to represent the indices of $x_i$’s partitions. For example, $p_i = j$ represents the $j^{th}$ partition of the $i^{th}$ feature. Thus, a stroke $x = (x_1, x_2, ..., x_d)$ can be written in integer form as $x = (p_1, p_2, ..., p_d)$. As expert hyper-rectangle $R^1$ is defined by the partition values, it can also be transformed to integer form as $R^1 = \{l_1 < p_1 \leq r_1, ..., l_d < p_d \leq r_d\}$, where $(l_i, r_i)$ defines the integer form.
of the value range in dimension $x_i$.

In Figure 5.1 suppose $x_1, x_2 \in [0, 1]$. Then, based on where it splits, $x_1$ has partitions: $[0, 0.3]^1$, $[0.3, 0.5]^2$ and $[0.5, 1]^3$ while $x_2$ has partitions: $[0, 0.2]^1$, $[0.2, 0.7]^2$ and $[0.7, 1]^3$. Thus, a stroke $x = (x_1 = 0.4, x_2 = 0.8)$ can be transformed into integer form as $x = (p_1 = 2, p_2 = 3)$ with $p_1$ and $p_2$ denoting the partition number for their respective features. The expert rectangle $R^1 = \{x_1 > 0.5, x_2 \leq 0.2\}$ can be transformed into $R^1 = \{2 < p_1 \leq 3, 0 < p_2 \leq 1\}$. As the RF grows each tree on a random subset of features, the number of features that defines a hyper-rectangle may be less than the total number of features.

With this transformation, a stroke is discretized to an integer point and an expert hyper-rectangle is discretized to a finite number of points. For example, $R^1 = \{1 < p_1 \leq 3, 2 < p_2 \leq 3\}$ is equivalent to $R^1 = \{p_1 \in \{2, 3\}, p_2 \in \{3\}\}$ denoting only two points in $R^1$: $(p_1 = 2, p_2 = 3)$ and $(p_1 = 3, p_2 = 3)$. As RF can deal with both numerical and categorical data, this transformation can be applied to arbitrary datasets. For simplicity, in the rest of this paper, “stroke” and “hyper-rectangle” denote the discretized integer forms.

5.2.3 Hyper-rectangle pruning

This process reduces the search space by 1) removing redundant expert hyper-rectangles, and 2) removing redundant points within the remaining hyper-rectangles so as to increase computational efficiency. We denote the set of hyper-rectangles that remain after this process by $R^*$. The RF feedback generation problem can now be solved approximately by finding a new stroke that is in the densest overlapping area of $R^*$’s.

Removing redundant expert hyper-rectangles: For an expert hyper-rectangle to be used to formulate feedback for a novice stroke $x$, it should contain at least one possible solution, i.e., one point that satisfies the cost constraint: $\exists x' \in R^1$ such that $\|x - x'\|_0 \leq 1$. When generating feedback for $x$, expert hyper-rectangles that cannot provide any possible solutions can be removed. For example, given $x = (p_1 = 2, p_2 = 3)$, the expert hyper-rectangle $R^1 = \{0 < p_1 \leq 1, 4 < p_2 \leq 5\}$ does not contain any feasible solutions as $x$ has to change two features ($p_1$ and $p_2$) to be moved into this hyper-rectangle.
5.2. RANDOM FOREST BASED FEEDBACK GENERATION

Pruning the remaining expert hyper-rectangles: Further, not all points in the remaining hyper-rectangle are feasible solutions. Consider the same novice stroke as above and $\mathcal{R}^1 = \{1 < p_1 \leq 3, 4 < p_2 \leq 5\}$. We can change $x$’s $p_2$ to 5 to move it into $\mathcal{R}^1$. However, there are other possible changes that can also move $x$ into $\mathcal{R}^1$, but require more than one feature change (e.g. changing $p_1$ to 3 and $p_2$ to 5 at the same time). Such changes violate the cost constraint, and as such can be pruned from the solution space by fixing the value of $p_i$ in $\mathcal{R}^1$ to its value in the novice stroke $x$, i.e., $\mathcal{R}^1 \rightarrow \mathcal{R}^* = \{p_1 = 2, 4 < p_2 \leq 5\}$.

5.2.4 Discrete approximation

Finding points that are in the densest overlapping area of $\mathcal{R}^*$’s can be done by iterating through all points in each $\mathcal{R}^*$. However, this approach is again computationally expensive. We avoid this through a discrete approximation (DA) method that uniformly selects a small number of points as representatives for $\mathcal{R}^*$, and takes the center of the densest area of these representatives as an approximation of the optimal solution.

We introduce a parameter $\alpha \in (0, 1]$ to indicate the proportion of representative values selected from each dimension of a hyper-rectangle $\mathcal{R}^* = \{l_1 < p_1 \leq r_1, l_2 < p_2 \leq r_2, \ldots, l_d < p_d \leq r_d\}$. The number of selected values from the $i^{th}$ dimension can be calculated by:

$$n_i = \begin{cases} r_i - l_i & \text{if } r_i - l_i \leq 2 \\ \lceil \alpha(r_i - l_i) \rceil + 2 & \text{otherwise} \end{cases} \quad (5.1)$$

For a dimension $p_i$ that contains only 1 or 2 values, we directly use these values to represent $p_i$. Otherwise, we divide $[l_i + 1, r_i]$ into $n_i + 1$ equal segments and take the values in the division positions with $l_i + 1$ and $r_i$ as representative values of $p_i$. After extracting these values out of each dimension, we derive $\prod_{i=1}^d n_i$ number of points by taking all the possible combinations.

To find the center of the densest area with respect to the representative points, we consider an area defined by a hyper-sphere with radius $\gamma$. Then, based on a computed Euclidean distance matrix between all representative points, the
point which has the most neighbours with a distance less than or equal to $\gamma$ is selected. The selected point is transformed from integer form back to its original form by taking the corresponding original value in the partition position. The feedback is then constructed based on the original form.

Figure 5.2 illustrates an example where there are 3 expert rectangles (blue, green and purple) and 2 features: $x_1$ and $x_2$ ($p_1, p_2$ are the integer forms of $x_1, x_2$ respectively). When taking 3 values in each dimension to represent a rectangle, we get 9 representative points per rectangle. Thus, with a radius $\gamma = 4$, we can find the red center point $x'$ of the densest area (the red dashed circle) of representative points. As shown in the figure, $x'$ is also located in the best expert space (the grey area overlapped by all three expert rectangles) that $x$ can be changed to. If the original form of $x$ is $(x_1 = 0.9, x_2 = 0.2)$ and $x'$ is $(x_1 = 0.4, x_2 = 0.2)$, then the feedback for $x$ is “decrease $x_1$ to 0.4”. In a nutshell, this method utilises 9 points to approximate each rectangle and the red circle with a tunable radius to approximate the grey overlapping area. The center of the circle is used as an approximation of the solutions in the grey area. The complete DA algorithm is shown in Algorithm 5.1.

5.3 Experimental evaluation

We compared the performance of the proposed DA method with 5 other methods:

1. **Split Voting (SV):** is the state-of-the-art real-time feedback generation method for RF introduced in [236].

2. **Integer Linear Programming (ILP):** solves the RF feedback generation problem by transforming it to an integer linear programming problem [31].

3. **Random Selection (Rand-Rand):** randomly picks a feature from a novice stroke and selects a random partition of that feature as the feedback.

4. **Random Iteration (Rand-Iter):** randomly selects a feature and iteratively selects the best partition to change it to as the feedback [31].
5.3. EXPERIMENTAL EVALUATION

Algorithm 5.1 Discrete approximation algorithm

**Input:** \( x \): novice stroke, \( R^{(1)} \): expert hyper-rectangles, \( C \): cost limit, \( \alpha, \gamma \)

**Output:** Feedback: actions of feature changes

1. \( P \): point set
2. for \( R_i^{(1)} \) in \( R^{(1)} \) do
3. if \( \ell(x, R_i^{(1)}) \leq C \) then
4. \( V \) = value sets for each dimension
5. \( R^f = \text{prune}(R_i^{(1)}) \)
6. for \( l_i < p_i \leq r_i \) in \( R^f \) do
7. \( n_i = \lceil \alpha (l_i - r_i) \rceil \)
8. \( V_i = n_i \) equal division values of \( (l_i, r_i) \)
9. end for
10. add all possible points in \( V \) to \( P \)
11. end if
12. end for
13. \( M = \text{pairwiseManhattanDistance}(P) \)
14. \( x' = \text{findCenter}(M, \gamma) \)
15. actions \( \leftarrow x - x' \)
16. **Return:** actions

5. **Iterative Selection (Iter-Iter):** iteratively tests all partitions of each feature while keeping the other features fixed. The overall best combination of feature and partition is selected as the feedback \[31\].

5.3.1 Experimental settings

The parameters of the DA method were tuned on the training data using a grid search. The parameter values that showed the best results under the real-time efficiency requirement of 1 second (\( \alpha = 0.5 \) and \( \gamma = 2 \)) were chosen. The methods were evaluated based on the following 3 measures:

1. **Success Rate (SR):** As defined below, it is the percentage of novice strokes that are successfully changed to expert strokes. Higher values of SR denote better performance.

\[
SR = \frac{\left| \{ x' | H(x') > 0.5 \} \right|}{\left| \{ x \} \right|} \tag{5.2}
\]
2. **Effectiveness (EFF):** As defined below, it is the value of the objective function defined in Problem 2.3.1, i.e., the probability of the target stroke being an expert stroke. It measures how effective the feedback is when applied to change the novice stroke. Higher values signify better effectiveness.

\[
EFF = P(y = 1|x') = H(x')
\]  

(5.3)

3. **Time-cost (TC):** Time (seconds) spent to formulate one feedback, lower is better. The lower the time-cost, the higher the efficiency.

Experiments were carried out on a typical PC with a 2.40GHz CPU. The ILP solver used for the ILP method was CPLEX\[1\] as recommended by the authors [31]. Our dataset consisted of 60K strokes (28K expert strokes and 32K novice strokes) performed by 7 experts and 12 novices. All methods were evaluated on a RF with 100 trees and a maximum depth of 5. A 12-fold leave-one-novice-out cross-validation was used to obtain an unbiased measure. In each fold, we used all strokes performed by one novice as the test set and trained a RF prediction model on the remaining strokes. All methods were then applied to formulate feedback for strokes in the test set using the trained RF. This design simulates the real-world scenario of an unknown novice using the simulator.

### 5.3.2 Results

As shown in Table 5.1, in terms of success rate and effectiveness, the proposed method (DA) is comparable to Iter-Iter and ILP that find optimal solutions and outperforms the other methods by a large margin. However, Iter-Iter and ILP take more than 10 seconds to formulate one feedback, and therefore fail to meet the real-time efficiency requirement of 1 second. Overall, DA achieves near-optimal success rate and effectiveness, and is highly efficient, indicating that it finds the ideal balance for real-time use in TBS simulation.

---

\[1\]https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer
5.3. EXPERIMENTAL EVALUATION

Table 5.1: Performance (mean±std) comparison between DA and other methods. The best results are highlighted in **bold**.

<table>
<thead>
<tr>
<th></th>
<th>success rate</th>
<th>effectiveness</th>
<th>time-cost (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rand-Rand</td>
<td>0.21±0.04</td>
<td>0.18±0.23</td>
<td>0.00±0.00</td>
</tr>
<tr>
<td>Rand-Iter</td>
<td>0.36±0.05</td>
<td>0.40±0.30</td>
<td>0.36±0.05</td>
</tr>
<tr>
<td>Iter-Iter</td>
<td><strong>0.89±0.00</strong></td>
<td><strong>0.87±0.06</strong></td>
<td>12.17±0.14</td>
</tr>
<tr>
<td>ILP</td>
<td>0.89±0.00</td>
<td>0.87±0.06</td>
<td>32.07±2.57</td>
</tr>
<tr>
<td>SV</td>
<td>0.60±0.05</td>
<td>0.65±0.33</td>
<td>0.02±0.00</td>
</tr>
<tr>
<td>DA</td>
<td><strong>0.89±0.00</strong></td>
<td>0.84±0.08</td>
<td>0.26±0.15</td>
</tr>
</tbody>
</table>

Figure 5.3: Performance of DA w.r.t. $\alpha$.

5.3.3 Parameter and scalability analysis

As shown in Figure 5.3, the time-cost of the proposed method DA increases consistently with the increase of the parameter $\alpha$. However, the effectiveness is stabilized after $\alpha = 0.5$. As $\alpha$ controls the number of representative points, this indicates that using more points for the approximation is not necessary and the effectiveness-efficiency balance can be achieved by the use of a smaller $\alpha$. We also tested the scalability of all methods with regard to the number of trees in the RF. As shown in Figure 5.4, the time-cost of ILP increases dramatically as the number of trees increases and it takes more than 4 minutes to formulate one
feedback using a RF with 1,000 trees. Iter-Iter is also seen to have a considerable increase in time-cost with the increase of the number of trees. However, the other methods, including the proposed method (DA), are more stable and remain highly efficient for large scale RFs.

5.4 Chapter summary

In this chapter, we further explored the use of adversarial techniques with traditional machine learning model RFs for feedback generation. We discussed the technical challenge in feedback generation from RFs, and proposed an novel approximation method inspired by the idea of adversarial perturbation. We discretized the hyper-rectangles of a random forest into integer form and proposed a novel method to formulate feedback using these hyper-rectangles. We also showed that the proposed method outperformed the state-of-the-art methods in terms of success rate and effectiveness while remaining highly efficient. Moreover, it is consistently efficient for large-scale random forests. The proposed method was tested in a specific application of virtual reality temporal bone surgery simulation, and can be generalized to other simulation training platforms where real-time feedback is of importance. Our work also confirms that
RF-based adversarial examples can be generated using the proposed method.
5. ADVERSARIAL FEEDBACK GENERATION WITH RANDOM FORESTS
Chapter 6

Dimensionality-Driven Learning with Noisy Labels

In this Chapter, we study the vulnerability of deep networks to noisy (incorrect) labels. Datasets with significant proportions of noisy (incorrect) class labels present challenges for training accurate DNNs. We propose a new perspective for understanding DNN generalization on such datasets, by investigating the dimensionality of the deep representation subspaces of training samples. We show that from a dimensionality perspective, DNNs exhibit quite distinctive learning styles when trained with clean labels versus when trained with a proportion of noisy labels. Based on this finding, we develop a new dimensionality-driven learning strategy, which monitors the dimensionality of subspaces during training and adapts the loss function accordingly. We empirically demonstrate that our approach is highly tolerant to significant proportions of noisy labels, and can effectively learn low-dimensional local subspaces that capture the data distribution.¹

¹This chapter of thesis is based on the following published paper:
6.1 Introduction

Deep networks (or DNNs) have demonstrated excellent performance in solving many complex problems, and have been widely employed for tasks such as speech recognition [74], computer vision [70] and gaming agents [185]. DNNs are capable of learning very complex functions, and can generalize well even for a huge number of parameters [144]. However, recent studies have shown that DNNs may generalize poorly for datasets which contain a high proportion of noisy (incorrect) class labels [233]. It is important to gain a fuller understanding of this phenomenon, with a view to development of new training methods that can achieve good generalization performance in the presence of variable amounts of label noise.

One simple approach for noisy labels is to ask a domain expert to relabel or remove suspect samples in a preprocessing stage. However, this is infeasible for large datasets and also runs the risk of removing crucial samples. An alternative is to correct noisy labels to their true labels via a clean label inference step [90, 117, 208, 209]. Such methods often assume the availability of a supplementary labelled dataset containing pre-identified noisy labels which are used to develop a model of the label noise. However, their effectiveness is tied to the assumption that the data follow the noise model. A different approach to tackle noisy labels is to utilize correction methods such as loss correction [57, 157], label correction [163], or additional linear correction layers [60, 194].

In this chapter, we first investigate the dimensionality of the deep representation subspaces learned by a DNN and provide a dimensionality-driven explanation of DNN generalization behavior in the presence of (class) label noise. Our analysis employs a dimensionality measure called Local Intrinsic Dimensionality (LID) [80, 81], applied to the deep representation subspaces of training examples. We show that DNNs follow two-stage of learning in this scenario: 1) an early stage of dimensionality compression, that models low-dimensional subspaces that closely match the underlying data distribution, and 2) a later stage of dimensionality expansion, that steadily increases subspace dimensionality in order to overfit noisy labels. This second stage appears to be a key factor behind the poor generalization performance of DNNs for noisy labels. Based on
this finding, we propose a new training strategy, termed *Dimensionality-Driven Learning*, that avoids the dimensionality expansion stage of learning by adapting the loss function.

## 6.2 Dimensionality of deep representation subspaces

We now briefly introduce the LID measure for assessing the dimensionality of data subspaces residing in the deep representation space of DNNs. We then connect dimensionality theory with the learning process of DNNs.

### 6.2.1 Local Intrinsic Dimensionality and its estimation

As introduced in Section 3.2, Local Intrinsic Dimensionality is an expansion-based measure of intrinsic dimensionality of the underlying data submanifold [81]. In the theory of intrinsic dimensionality, classical expansion models (such as the expansion dimension [93] and generalized expansion dimension [83]) measure the rate of growth in the number of data objects encountered as the distance from the reference sample increases. Intuitively, in Euclidean space, the volume of an $D$-dimensional ball grows proportionally to $r^D$ when its size is scaled by a factor of $r$. From the above rate of volume growth with distance, the dimension $D$ can be deduced from two volume measurements as:

$$
\frac{V_2}{V_1} = \left(\frac{r_2}{r_1}\right)^D \Rightarrow D = \frac{\ln(V_2/V_1)}{\ln(\frac{r_2}{r_1})}. \quad (6.1)
$$

The aforementioned expansion-based measures of intrinsic dimensionality would determine $D$ by estimating the volumes in terms of the numbers of data points captured by the balls. Transferring the concept of expansion dimension from the Euclidean space to the statistical setting of continuous distance distributions, the notion of ball volume is replaced by the probability measure associated with the balls. This leads to the formal definition of LID [81], as in Definition 2.

The LID score of a reference sample can be estimated by its $k$-nearest neighbours from the training set, as in Equation (3.4). Since computing neighborhoods with respect to the entire dataset $X$ can be prohibitively expensive, we
6. DIMENSIONALITY-DRIVEN LEARNING WITH NOISY LABELS

Figure 6.1: The subspace dimensionality (average LID scores) and train/test accuracy throughout training for a 12-layer CNN on CIFAR-10 dataset with clean (left subfigure) and noisy labels (right subfigure). The average LID scores were computed at layer 11.

will estimate LID of a training example $x$ from its $k$-nearest neighbor set within a batch randomly selected from $X \sim \mathcal{P}$. Consider a $L$-layer neural network $F: \mathcal{P} \rightarrow \mathbb{R}^C$, where $f^i$ is the intermediate transformation of the $i$-th layer, and $C$ is a positive number indicating the number of classes. Given a batch of training samples $X_B \subseteq X$, and a reference point $x \sim \mathcal{P}$ (not necessarily a training sample), we estimate the LID score of $x$ as:

$$
\hat{\text{LID}}(x, X_B) = -\left(\frac{1}{k} \sum_{i=1}^{k} \log \frac{r_i(g(x), g(X_B))}{r_{\max}(g(x), g(X_B))}\right)^{-1},
$$

(6.2)

where $g = h^{L-1}$ is the output of the second-to-last layer of the network, $r_i(g(x), g(X_B))$ is the distance of $g(x)$ to its $i$-th nearest neighbor in the transformed set $g(X_B)$, and $r_{\max}$ represents the radius of the neighborhood. $\hat{\text{LID}}(x, X_B)$ reveals the dimensional complexity of the local subspace in the vicinity of $x$, taken after transformation by $g$. Provided that the batch is chosen sufficiently large so as to ensure that the $k$-nearest neighbor sets remain in the vicinity of $g(x)$, the estimate of LID at $g(x)$ within the batch serves as an approximation to the value that would have been computed within the full dataset $g(x)$. 

108
6.2. DIMENSIONALITY OF DEEP REPRESENTATION SUBSPACES

Figure 6.2: The subspace dimensionality (average LID scores) and train/test accuracy throughout training for a 5-layer CNN on MNIST dataset with clean (left subfigure) and noisy labels (right subfigure). The average LID scores were computed at layer 4.

6.2.2 Subspace dimensionality and noisy labels

We now show by means of an example how the subspace dimensionality of training and test examples is affected by the quality of label information, as the number of training epochs is increased. For our example, we trained a 5-layer CNN (convolutional neural network) on MNIST (an image data set with 10 categories of handwritten digits [110]) and a 12-layer CNN on CIFAR-10 (a natural image data set with 10 categories [100]) using SGD, cross-entropy loss, and two different label quality settings: (1) clean labels for all training samples; (2) noisy labels for 40% of the training samples, generated by uniformly and randomly replacing the correct label with one of the 9 incorrect labels. LID values at layer 4 for MNIST and layer 11 for CIFAR-10 were averaged over 10 batches of 128 points each, for a total of 1280 test points. The resulting LID scores and the train/test accuracies are shown in Figures 6.1 and 6.2. When learning with clean labels, we observe a decreasing trend in LID score and an increasing trend in accuracy as the number of training epochs increases. However, when learning with noisy labels, we see a very different trend: first a decrease in LID followed by an increase, accompanied by an initial increase in test accuracy followed by a decrease. We observed similar dimensionality trends for a 6-layer CNN on SVHN [143] and a 44-layer ResNet [70] on CIFAR-100 [100].
Clearly, in these two situations, the DNNs are exhibiting different learning styles. For training data with clean labels, the network gradually transforms the data to subspaces of low dimensionality. Once the subspaces of the lowest dimensionality has been found, the network effectively stops learning: the test accuracy stabilizes at its highest level and the dimensionality stabilizes at its lowest. On the other hand, for training data with noisy labels, the network initially learns a transformation of the data to subspaces of lower dimensionality, although not as low as when training on data with clean labels. Thereafter, the network progressively attempts to accommodate noisy labels by increasing the subspace dimensionality.

6.2.3 Two-Stage of learning of DNNs on noisy labels

From the above empirical results, we find that DNNs follow two-stage of learning in the presence of label noise: 1) an early stage of dimensionality compression, in which the dimensionalities associated with the underlying data manifold are learned; and 2) a later stage of dimensionality expansion, in which the subspace dimensionalities steadily increase as the learning process overfits to the noisy data.

One possible explanation for this phenomenon can be found in the effect of transformation on the neighborhood set of test points. Given a training point \( x \in X \), its initial spatial location (before learning) would relate to a low-dimensional local subspace determined by the underlying manifold (call this subspace \( A \)). Although the initial neighborhood of \( x \) would likely contain many data points that are also close to manifold \( A \), the LID estimate would not necessarily be the exact dimension of \( A \). LID reveals the growth characteristics of the distance distribution from \( x \), which is influenced by — but not equal to — the dimension of the manifold to which \( x \) is best associated.

As the learning process progresses, the manifold undergoes a transformation by which it progressively achieves a better fit to the training data. If \( x \) is labeled correctly, and if many of its neighbors also have clean labels, the learning process can be expected to converge towards a local subspace of relatively low intrinsic dimensionality (as observed in the left-hand plot of Figures 6.1 and
6.2. DIMENSIONALITY OF DEEP REPRESENTATION SUBSPACES

However, it should be noted that the learning process still risks overfitting to the data, if carried out too long. With overfitting, the dimensionality of the local manifold would be expected to rise eventually.

If x is incorrectly labeled, each epoch in the learning process progressively causes x — or more precisely, its transform (call it \( x' \)) — to migrate to a new local subspace (call it \( A' \)) associated with members of the same label that was incorrectly applied to x. During this migration, the neighborhood of \( x' \) tends to contain more and more points of \( A' \) that share the same label as x, and fewer and fewer points from the original neighborhood in \( A \). With respect to the points of \( A' \), the mislabeled point \( x' \) is spatially an outlier, since its coordinates relate to \( A \) and not \( A' \); thus, the presence of \( x' \) forces the local subspace around it to become more high-dimensional in order to accommodate (or compress) it. This distortion results in a dimensionality expansion in the vicinity of \( x' \) that would be expected to be reflected in LID estimates based at \( x' \). Stopping the learning process earlier allows \( x' \) to find its neighborhood in \( A \) before the local subspace is corrupted by too many neighbors from \( A' \), which thus leads to better learning of the true data distribution and improved generalization to test data.

This explanation of the effect of incorrect labeling in terms of local subspaces is consistent with our previous work in Chapter [3] for the effect of adversarial perturbation on DNN classification. In this situation, rather than directly assigning an incorrect label to the test item while leaving its spatial coordinates unchanged, the adversary must instead attempt to move a test point into a region associated with an incorrect class by means of an antagonistic learning process. In both cases, regardless of how the test point is modified, the neighborhoods of the transformed points are affected in a similar manner: as the neighborhood membership evolves, the local intrinsic dimensionality can be expected to rise. The associated changes in LID estimates have been used as the basis for the effective detection of a wide variety of adversarial attacks. Recent theoretical work for adversarial perturbation in nearest-neighbor classification further supports the relationship between LID and local transformation of data, by showing that the magnitude of the perturbation required in order to subvert the classification diminishes as the local intrinsic dimensionality and data sample size grow [2].
6.3 Dimensionality-Driven learning strategy

In the previous section, we observed that learning in the presence of noisy labels has two stages: dimensional compression, followed by dimensional expansion. Motivated by these observations, we propose a Dimensionality-Driven Learning (D2L) strategy whose objective is to avoid the overfitting and loss of test accuracy associated with dimensional expansion.

Given a training sample $x$, we denote its raw label as $y$ and its predicted label as $\hat{y}$, where both $y$ and $\hat{y}$ are ‘one-hot’ indicator vectors. $(\hat{\text{LID}}_0, \ldots, \hat{\text{LID}}_t, \ldots, \hat{\text{LID}}_T)$ is a sequence of LID scores, where $\hat{\text{LID}}_i$ represents the LID score computed from the second-to-last DNN layer at the $i$-th training epoch ($T$ epochs in total). Each LID score is produced as follows. $m$ batches of samples are randomly selected $X_B^1, \ldots, X_B^m$ and for each $X_B^i$ and each of its members $x$, $\text{LID}(x, X_B^i)$ is computed. This gives $m \times |X_B^i|$ LID estimates, which are then averaged to compute the LID score for the epoch (later, in the experiments, we use $m = 10$ and $|X_B^i| = 128$).

To avoid dimensionality expansion during training with noisy labels, we propose to reduce the effect of noisy labels on learning the true data distribution using the following adaptive LID-corrected labels:

$$y^* = \alpha_i y + (1 - \alpha_i)\hat{y}, \quad (6.3)$$

where $\alpha_i$ is a LID-based factor that updates at the $i$-th training epoch:

$$\alpha_i = \exp \left( -\lambda \frac{\hat{\text{LID}}_i}{\min_{j=0}^{i-1} \hat{\text{LID}}_j} \right), \quad (6.4)$$

where $\lambda = i/T$ is a weighting that indicates decreasing confidence in the raw labels when the training proceeds to the dimensionality expansion stage (that is, when LID begins to increase). The training loss can then be refined as:

$$\mathcal{L} = -\frac{1}{N} \sum_{n=1}^{N} \sum_{y_n} y_n^* \log P(y_n^*|x_n), \quad (6.5)$$

where $N$ is the total number of training samples and $P(y_n^*|x_n)$ is the predicted
class probability of $y^*_n$ given $x_n$.

Interpreting Equations (6.3) - (6.5), we can regard D2L as a simulated annealing algorithm that attempts to find an optimal trade-off between subspace dimensionality and prediction performance. The role of $\alpha$ is an exponential decay factor that allows for interpolation between raw and predicted label assignments according to the degree of dimensional expansion observed over the learning history. Here, dimensional expansion is assessed in terms of the ratio of two average LID scores: the score observed at the current epoch, and the lowest score encountered at earlier epochs. As the learning enters the dimensional expansion stage, this ratio exceeds 1, and the exponential decay factor begins to favor the current predicted label. The complete D2L learning strategy is shown in Algorithm 6.1. Note that the computational cost of LID estimation through batch sampling is low compared to the overall training time ($t_{\text{LID}}/t_{\text{training}} \approx 1 - 2\%$), as it requires only the pairwise distances within a few batches.

To identify the turning point between the two stages of learning, we employ an epoch window of size $w \in [1, T - 1]$ so as to allow $w$ epochs of initialization for the network, and to reduce the variation of stochastic optimization. The turning point is flagged when the LID score of the current epoch is two standard deviations higher than the mean LID score of the $w$ preceding epochs, until which the D2L loss is equivalent to the cross-entropy loss (enforced by setting $\alpha$ equal to 1). The epoch at which the turning point is identified can be regarded as the first epoch at which overfitting occurs; for this reason, we roll the model state back to that of the previous epoch, and begin the interpolation between the raw and predicted label assignments. Although we find in the experimental results of Section 6.4 that this strategy works consistently well for a variety of datasets, further variations upon this basic strategy may also be effective.

### 6.4 Experiments

We evaluate our proposed D2L learning strategy, comparing the performance of our model with state-of-the-art baselines for noisy label learning.
Algorithm 6.1 Dimensionality-Driven Learning (D2L)

Input:
- dataset $X$, network $F(x)$, total epochs $T$, epoch window $w$, number of batches for LID estimation $m$.

Initialization:
- epoch $i \leftarrow 0$, $lids \leftarrow [], \alpha_0 \leftarrow 1$, turning epoch $u \leftarrow -1$.

1: repeat
2:   Train $F(x)$ for one epoch.
3:   $lid \leftarrow 0$, $\lambda \leftarrow i/T$.
4:   for $j = 1$ to $m$ do
5:     Sample $X_B$ from $X$.
6:     $lid \leftarrow lid + \frac{1}{|X_B|} \sum_{k=1}^{|X_B|} \text{LID}(x, X_B)$.
7:   end for
8:   $lids[i] \leftarrow lid/m$.
9:   if $i \geq w$ and $u = -1$ and
10:      $lid - \text{mean}(lids[i-w : i-1]) > 2 \cdot \text{std}(lids[i-w : i-1])$ then
11:         $u \leftarrow i-1$. # turning point found
12:      Rollback $F(x)$ to the $u$-th epoch.
13:   end if
14:   if $u > -1$ then
15:      $\alpha_i = \exp(-\lambda \cdot lids[i]/\min(lids[0 : i-1]))$.
16:   else
17:      $\alpha_i = \alpha_0$
18:   end if
19:   $y^* = \alpha_i y + (1 - \alpha_i) \hat{y}$.
20:   Update loss to $L = -\frac{1}{N} \sum_{n=1}^{N} y_n^* \log P(y_n^*|x_n)$.
21:   $i \leftarrow i + 1$.
22: until $i = T$ or early stopping.

6.4.1 Empirical understanding of D2L

We first provide an empirical understanding of the proposed D2L learning strategy on subspace learning, hypothesis learning, representation learning and model analysis.

114
6.4. EXPERIMENTS

Figure 6.3: The trend of test accuracy and subspace dimensionality on CIFAR-10 with 40% noisy labels.

6.4.1.1 Experimental setup

The experiments were conducted on the benchmark dataset CIFAR-10 [100]. We used a 12-layer CNN architecture. All networks were trained using SGD with momentum 0.9, weight decay $10^{-4}$ and an initial learning rate of 0.1. The learning rate was divided by 10 after epochs 40 and 80 ($T = 120$ epochs in total). Simple data augmentations (width/height shift and horizontal flip) were applied. Noisy labels were generated by introducing symmetric noise, in which the labels of a given proportion of training samples are flipped to one of the other class label, selected with equal probability. In [208] this noisy label generation scheme has been verified to be more challenging than that of restricted (asymmetric) label noise, which assumes that mislabelling only occurs within a specific set of classes [163, 157].

6.4.1.2 Competing strategies

1) Backward [157]: training via loss correction by multiplying the cross-entropy loss by a noise-aware correction matrix; 2) Forward [157]: training with label correction by multiplying the network prediction by a noise-aware correction matrix; 3) Boot-hard [163]: training with new labels generated by a convex combination (the “hard” version) of the noisy labels and their predicted labels; 4) Boot-soft [163]: training with new labels generated by a convex combina-
6. DIMENSIONALITY-DRIVEN LEARNING WITH NOISY LABELS

Figure 6.4: The trend of test accuracy and subspace dimensionality on CIFAR-10 with 60% noisy labels.

The parameters of the competitors were configured according to their original papers. For our proposed D2L, we set $k = 20$ for LID estimation, and used the average LID score over $m = 10$ random batches of training samples as the overall dimensionality of the representation subspaces.

6.4.1.3 Effect on subspace learning

We illustrate the effect of D2L on subspace learning by investigating the dimensionality (measured by LID) of the deep representation subspaces learned by DNNs and the test accuracy throughout training. The results are presented in Figure 6.3 and 6.4 for the CIFAR-10 dataset, with noisy label proportions set to 40% and to 60%. First, examining the test accuracy (the left-hand plots), we see that D2L can stabilize the test accuracy after around 60 epochs regardless of the noise rate, whereas the competitors experience a substantial decrease in test accuracy. This indicates the effectiveness of D2L in limiting the overfitting to noisy labels. Second, we focus on the dimensionality of the representation subspaces learned by different models (the right-hand plots). We observe that D2L is capable of learning representation subspaces which have significantly lower dimensionality than other models. It can also be noted that lower-dimensional subspaces lead to better generalization and higher test accuracy. This supports
6.4. EXPERIMENTS

Figure 6.5: The hypothesis complexity (measured by CSR) on CIFAR-10 with 40% (left) and 60% (right) noisy labels.

...our claim that the true data distribution is of low dimensionality, and that D2L is capable of learning the low-dimensional true data distribution even with a large proportion of noisy labels. Note that for the case of 60% label noise, the low test accuracy of the ‘backward’ model, as well as the low dimensionality of the learned subspaces, together show that this competitor suffered from underfitting.

6.4.1.4 Effect on hypothesis learning

We investigate the complexity of the hypotheses learned from different models. Given a hypothesis space \( \mathcal{H} \), a learned hypothesis \( h \in \mathcal{H} \) from a DNN with lower complexity is expected to generalize better. Here, we use the recently proposed Critical Sample Ratio (CSR) [4] as the measure for hypothesis complexity. CSR measures the density around the decision boundaries, where a high CSR score indicates a complex decision boundary and hypothesis.

As shown in Figure 6.5, the complexity of the learned hypothesis from D2L is significantly lower than that of its competitors. Recalling the results from Figure 6.3 and 6.4, where D2L achieved the highest test accuracy, we conclude that a simpler hypothesis does lead to better generalization, and that D2L is capable here of learning smoother decision boundaries and a simpler hypothesis than its competitors.
6. DIMENSIONALITY-DRIVEN LEARNING WITH NOISY LABELS

Figure 6.6: Representations (t-SNE 2D embeddings) of two CIFAR-10 classes, ‘airplane’ (A) and ‘cat’ (B), learned by cross-entropy (left) and our D2L model (right), with 60% of the class labels set to noise.

6.4.1.5 Effect on representation learning

To analyze the effectiveness of D2L for representation learning, we visualize dataset representations in 2-dimensional embeddings using t-SNE [128], a commonly-used dimensionality reduction technique for the visualization of high-dimensional data [107]. Figure 6.6 presents the reduced 2D embeddings of 500 randomly selected samples from each of two classes on CIFAR-10. For each class, 40% of the samples were assigned correct labels (the ‘clean’ samples), and 60% were assigned incorrect labels chosen uniformly at random from the 9 other classes (the ‘noisy’ samples). We see that D2L (the right-hand plot) can learn high-quality representations that accurately separate the two classes of objects (blue vs red), and can effectively isolate noisy samples (magenta/cyan) from clean samples (blue/red). However, for both classes, representations learned by cross-entropy (the left-hand plot) suffer from significant overlapping between clean and noisy samples. Note that the representations of noisy samples learned by D2L are more fragmented, since the noisy labels are from many different classes. Overall, D2L is able to learn a high-quality representation from noisy datasets.
6.4. EXPERIMENTS

Figure 6.7: Grid searching neighborhood size \( k \) (left) and number of batches \( m \) (right) for the estimation of LID on CIFAR-10 with various noise rate.

### 6.4.1.6 Parameter sensitivity

We assess the sensitivity of D2L to the neighborhood size \( k \) and the number of batches \( m \) used to compute the mean LID. Figure 6.7 shows that D2L is relatively insensitive to these two hyper-parameters on the CIFAR-10 dataset. We observed similar behavior with the other three datasets.

### 6.4.2 Robustness against noisy labels

Finally, we evaluate the robustness of D2L against noisy labels under varying noise rates (0%, 20%, 40%, and 60%) on several benchmark datasets, comparing to state-of-the-art baselines for noisy label learning. Here, we focus on the symmetric noise setting, and will study other types of label noise such as asymmetric label noise, open-set label noise [213] and adversarial label noise [63, 202] in future research.

#### 6.4.2.1 Experimental setup

Experiments were conducted on several benchmark datasets: MNIST [110], SVHN [143], CIFAR-10 [100] and CIFAR-100 [100]. We used a LeNet-5 network [110] for MNIST, a 6-layer CNN for SVHN, a 12-layer CNN for CIFAR-10 and a ResNet-44 network [70] for CIFAR-100. All networks were trained using SGD with momentum 0.9, weight decay \( 10^{-4} \) and an initial learning rate of 0.1. The learning rate is divided by 10 after epochs 20 and 40 for MNIST/SVHN (50
6. DIMENSIONALITY-DRIVEN LEARNING WITH NOISY LABELS

Table 6.1: Test accuracy (%) of different models on MNIST with varying noise rates (0% – 60%). The mean accuracy (±std) over 5 repetitions of the experiments are reported, and the best results are highlighted in **bold**.

<table>
<thead>
<tr>
<th>Model / Noise Rate</th>
<th>MNIST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
</tr>
<tr>
<td>cross-entropy</td>
<td>99.24±0.0</td>
</tr>
<tr>
<td>forward</td>
<td>99.30±0.0</td>
</tr>
<tr>
<td>backward</td>
<td>99.23±0.1</td>
</tr>
<tr>
<td>boot-hard</td>
<td>99.13±0.2</td>
</tr>
<tr>
<td>boot-soft</td>
<td>99.20±0.0</td>
</tr>
<tr>
<td>D2L</td>
<td>99.28±0.0</td>
</tr>
</tbody>
</table>

...}

epochs in total), after epochs 40 and 80 for CIFAR-10 (120 epochs in total), and after epochs 80, 120 and 160 for CIFAR-100 (200 epochs in total) [84]. Simple data augmentations (width/height shift and horizontal flip) were applied on CIFAR-10 and CIFAR-100. Noisy labels were generated as described in Section 6.4.1. On a particular dataset, the compared methods differ only in their loss functions — they share the same CNN architecture, regularizations (batch normalization and max pooling), and the number of training epochs. We repeated the experiments 5 times with different random seeds for network initialization and label noise generation.

6.4.2.2 Results

We report the mean test accuracy and standard deviation over 5 repetitions of the experiments in Table 6.1-6.4. D2L outperforms its competitors consistently across all datasets and across all noise rates tested. In particular, the performance gap between D2L and its competitors increases as the noise rate is increased from 20% to 60%. We also note that as the noise rate increases, the accuracy drop of D2L is the smallest among all models. Even with 60% label noise, D2L can still obtain a relatively high classification accuracy, which indicates that D2L may have the potential to be an effective strategy for semi-supervised learning.
6.5 CHAPTER SUMMARY

Table 6.2: Test accuracy (%) of different models on SVHN with varying noise rates (0% – 60%). The mean accuracy (±std) over 5 repetitions of the experiments are reported, and the best results are highlighted in **bold**.

<table>
<thead>
<tr>
<th>Model / Noise Rate</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
</tr>
<tr>
<td>cross-entropy</td>
<td>90.12±0.0</td>
</tr>
<tr>
<td>forward</td>
<td>90.22±0.1</td>
</tr>
<tr>
<td>backward</td>
<td>90.16±0.1</td>
</tr>
<tr>
<td>boot-hard</td>
<td>89.47±0.0</td>
</tr>
<tr>
<td>boot-soft</td>
<td>89.26±0.0</td>
</tr>
<tr>
<td>D2L</td>
<td><strong>90.32±0.0</strong></td>
</tr>
</tbody>
</table>

Table 6.3: Test accuracy (%) of different models on CIFAR-10 with varying noise rates (0% – 60%). The mean accuracy (±std) over 5 repetitions of the experiments are reported, and the best results are highlighted in **bold**.

<table>
<thead>
<tr>
<th>Model / Noise Rate</th>
<th>CIFAR-10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
</tr>
<tr>
<td>cross-entropy</td>
<td>89.31±0.1</td>
</tr>
<tr>
<td>forward</td>
<td><strong>90.27±0.1</strong></td>
</tr>
<tr>
<td>backward</td>
<td>89.03±0.3</td>
</tr>
<tr>
<td>boot-hard</td>
<td>89.06±0.3</td>
</tr>
<tr>
<td>boot-soft</td>
<td>89.46±0.2</td>
</tr>
<tr>
<td>D2L</td>
<td>89.41±0.2</td>
</tr>
</tbody>
</table>

6.5 Chapter summary

In this chapter, we have explored the vulnerability of DNNs to noisy labels. In particular, we investigated the generalization behavior of DNNs for noisy labels in terms of the intrinsic dimensionality of local subspaces. We observed that dimensional compression occurs early in the learning process, followed by dimensional expansion as the process begins to overfit. Employing a simple measure of local intrinsic dimensionality (LID), we proposed a Dimensionality-Driven Learning (D2L) strategy for avoiding overfitting that identifies the learning epoch at which the transition from dimensional compression to dimensional
Table 6.4: Test accuracy (%) of different models on CIFAR-100 with varying noise rates (0% – 60%). The mean accuracy (±std) over 5 repetitions of the experiments are reported, and the best results are highlighted in **bold**.

<table>
<thead>
<tr>
<th>Model / Noise Rate</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
</tr>
<tr>
<td>cross-entropy</td>
<td>68.20±0.2</td>
</tr>
<tr>
<td>forward</td>
<td>68.54±0.3</td>
</tr>
<tr>
<td>backward</td>
<td>68.48±0.3</td>
</tr>
<tr>
<td>boot-hard</td>
<td>68.31±0.2</td>
</tr>
<tr>
<td>boot-soft</td>
<td>67.89±0.2</td>
</tr>
<tr>
<td>D2L</td>
<td><strong>68.60±0.3</strong></td>
</tr>
</tbody>
</table>

expansion occurs, and then suppresses the subsequent dimensionality expansion. D2L delivers very strong classification performance across a range of scenarios with high proportions of noisy labels.
Chapter 7

Conclusion and Future Works

In this thesis, we investigated the vulnerability of deep networks to adversarial perturbations and noisy labels. We obtained new understandings of DNNs and their learning behavior by investigating the dimensionality of data subspaces learned by the networks. Based on these new understandings, we proposed to characterize the dimensional properties of local adversarial subspaces in the vicinity of adversarial examples, and use these properties to discriminate adversarial examples. We also developed a novel learning strategy (e.g. D2L) to identify and avoid subspace dimensionality expansion, so as to robustly train DNNs in the presence of noisy labels. We also explored the possibility of using adversarial techniques in a beneficial way, specifically for the generation of performance feedback in simulation based learning. In this section, we first summarize the contributions of this thesis and their impact. Then we discuss some limitations of the work and potential directions for future work.

7.1 Summary of contributions

Firstly, in order to investigate the vulnerability of DNNs to adversarial perturbations, we explored the dimensional properties of adversarial subspaces immediately surrounding adversarial examples, using the Local Intrinsic Dimensionality (LID) measure. We found that adversarial subspaces are of higher intrinsic dimensionality than normal data subspaces. We provided a theore-
7. CONCLUSION AND FUTURE WORKS

A critical explanation to this high dimensionality phenomenon of adversarial subspaces: adversarial perturbations need to escape a low dimensional manifold to a subspace of much higher dimensionality so as to transition from one class to another. We demonstrated that this characteristic can be used to effectively discriminate adversarial examples generated using state-of-the-art attacking methods. Our analysis of the LID characteristic for adversarial subspaces not only motivates new directions of effective adversarial defense, but also opens up more challenges for developing new attacks to better understand the vulnerabilities of DNNs.

Secondly, we explored the possibilities of using adversarial techniques to generate feedback for intelligent tutoring in simulation-based learning (SBL). In contrast to the multitude of work that exploit adversarial techniques in order to attack deep networks, we demonstrated, that such techniques can also be utilized in a beneficial way. Our work proved that, with proper constraints, knowledge can be extracted efficiently, with a high level of accuracy, from machine learning models to support long-term learning goals.

Thirdly, we demonstrated that feedback perturbations can also be generated from random forest (RF) models. We viewed the decision space of a RF as overlapping hyper-rectangles and feedback perturbations as spatial transitions that can move a sample from ‘novice’ space to ‘expert’ space. Based on this geometric view of a RF, we proposed a novel generation method with a series of techniques including random forest discretization, hyper-rectangle pruning and discrete approximation of the pruned hyper-rectangles, to find the optimal path for class transition. We demonstrated that this method was not only accurate, but was also highly efficient. This work contributes to the knowledge of feedback generation in SBL and also motivates new adversarial attacks on RF models.

Finally, we investigated DNNs and their vulnerability in learning with noisy labels. We analyzed the intrinsic dimensionality of representation spaces learned by DNNs, and monitored the learning process with clean labels versus noisy labels. We found that DNNs exhibit different learning styles in clean learning when compared to noisy learning. Clean learning is associated with dimensionality compression that tends to map data to a simpler manifold that best approx-
imates the underlying data structure, while learning with noisy labels is associated with dimensionality shift from compression to expansion to accommodate noisy labels. Based on this observation, we proposed a dimensionality-driven learning (D2L) strategy to avoid dimensionality expansion and overfitting to noisy labels. We demonstrated that the proposed D2L learning strategy can learn low-dimensional subspaces, simple hypothesis and high-quality representations. This part of our work not only sheds new light on DNN learning behavior under label noise, but also motivates new dimensionality-based exploration of DNN generalization in general.

7.2 Thesis limitations and future work

Although we conducted empirical analysis on the intrinsic dimensionality of DNN subspaces in Chapters 3 and 6, and have found significant patterns, theoretical formulation of DNN subspace dimensionality has not yet been developed. Such a formulation can reveal the underlying nature of deep networks, and motivate new DNN architectures as well as more robust learning strategies. We believe that dimensionality-based analysis opens up new directions for understanding and enhancing the behavior of DNNs.

In Chapter 3, we investigated discriminating adversarial examples using subspace dimensionality measured by LID. In the learning process, the activation values at each layer of a DNN can be regarded as a transformation of the input to a space in which the LID values have themselves also been transformed. A full understanding of LID characteristics of DNN space should take into account the effect of DNN transformations on these characteristics. This is a challenging question, since it requires a better understanding of the DNN learning processes themselves. One possible avenue for future research may be to model the dimensional characteristics of the DNN itself, and to empirically verify how they influence the robustness of DNNs to adversarial attacks. This direction of research could motivate novel adversarial defenses to secure DNNs.

In addition to theoretical formulation and understanding of DNN subspace dimensionality, another open issue for future research is the empirical investi-
7. CONCLUSION AND FUTURE WORKS

gation of the effect of LID estimation quality on the performance of adversarial detection. As evidenced by the improvement in performance observed when increasing the minibatch size in Chapter 3 (see Figure 3.5), it stands to reason that improvements in estimator quality or sampling strategies could both be beneficial in practice. Improved estimators that can produce more stable and accurate estimation of subspace dimensionality is also an interesting direction for future work.

LID-based adversarial defense is another interesting topic to investigate. Although we found, in our experiments in Chapter 3, that adversarial examples are of higher subspace dimensionality, an adversarial defense method that can make use of such characteristic to secure DNNs is still underdevelopment. One possible future work in this direction is to develop an optimization based method to reverse the perturbation of an adversarial example by pulling the example back to normal data submanifold that of lower dimensionality. And this can be done by gradient descent based optimization method at deeper layers of the network. The reversed adversarial example can then be fed into the network for prediction. Although generative adversarial networks (GANs) have been demonstrated to be powerful generative models, the properties of GAN generated images are still not clear. It would be interesting to see if GANs can generate images from high dimensional subspaces and whether they can be used as adversarial examples to attack DNNs.

In Chapter 6, we studied the learning behaviors of DNNs in the presence of noisy labels. However, DNN learning is a complex process associated with training data, DNN architecture and regularizations. Theoretical formulation of DNN subspace dimensionality, and investigation of the effects of data augmentation, DNN architecture and regularization techniques such as batch normalization [87] and dropout [188] are possible directions for future research.

It is interesting, from the dimensionality perspective, to see how these techniques work individually and also in combinations to improve learning from data. For example, from the dimensionality perspective, to understand the effect of residual connections in the residual network as to why such connections can significant improve generalization performance. It is also interesting to investigate how data augmentation contributes to the learning of DNNs and
7.2. THESIS LIMITATIONS AND FUTURE WORK

whether the augmentation can be improved by generating samples from only low-dimensional submanifolds. The effect of batch normalization should also be reflected by the LID characteristics of the regularized space. Similar analysis can also be applied to regularization techniques such as weight regularization and dropout. New findings in these directions of research could eventually lead to better learning architectures, improved denoising techniques, more robust learning models, or a better understanding of how these techniques can contribute to the generalization capacity of DNNs.

Our work on noisy label learning in Chapter 6 only investigated symmetric label noise in a closed-set setting, however, there also exist other types of label noise such as asymmetric label noise and open-set label noise. It is an interesting future work to investigate how DNN learns under these different settings of label noise, from the perspective of subspace dimensionality. Such an investigation could motivate new learning strategies that allow DNNs to learn adaptively from data with mixed types of label noise. In addition to noisy labels, another open issue is the investigation of how other forms of input noise such as adversarial or corrupted input noise can affect local subspace dimensionality and DNN learning behavior. And such investigations can be done not only in supervised learning but also in unsupervised learning.

Further investigation of the dimensional properties of deep learning models in semi-supervised or unsupervised learning settings may also be worthwhile. Such an understanding would be of great potential for a wide range of applications such as denoising techniques, semi-supervised learning and domain adaption, as well as learning with novelty including zero-shot and one-shot learning. Dimensionality-based investigation can also be applied to study GAN generated images. For instance, study the LID characteristic of generated images and its correlation to inception score (a popular quality metric for GANs), in measuring the quality of an image. It is also worthwhile to explore LID-guided generation of images using GANs, where images can be enforced to be generated from low-dimensional submanifolds. In GANs, the generator is to learn the real data distribution, in an adversarial fashion with the discriminator, so as to generate “real” samples. It can be interesting to see how the manifold of generated data distribution matches the manifold of real data distribution.
These future work may motivate improved GANs.

In Chapters 4 and 5 we developed effective and efficient feedback generation methods for SBL. However, improving human performance in practice is a very challenging task. It involves many aspects of the learning process such as the learning environment, task complexity, knowledge level of the learner, and feedback intervention. Therefore, whether or not and how these feedback generation methods improve learning in practical applications is worth further investigation. User studies can be conducted to perform such an investigation. Meanwhile, it is interesting to explore other forms of feedback with the technique of deep learning. For example, convolutional neural networks and recurrent neural networks can be used to predict the next most effective moves in a certain task, based on the 3D scenes of the simulated environment. Such generated movements can be delivered to the trainee as feedback to support skill learning and decision making. It is also interesting to explore how reinforcement learning techniques can be used in simulated learning environments to teach a machine to perform complex tasks such as surgery and flying a plane automatically without human intervention.
Bibliography


