Understanding Deep Neural Networks from the Perspective of Piecewise Linear Property

by

Xia Hu

B.Sc., University of Science and Technology of China, 2013

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2021

Declaration of Committee

Name: **Xia Hu**

Degree: **Doctor of Philosophy**

Thesis title: **Understanding Deep Neural Networks from the Perspective of Piecewise Linear Property**

Committee:

Chair: Jiannan Wang
Associate Professor, Computing Science

Jian Pei
Supervisor
Professor, Computing Science

Oliver Schulte
Committee Member
Professor, Computing Science

Mo Chen
Examiner
Assistant Professor, Computing Science

Jingrui He
External Examiner
Associate Professor
School of Information Sciences
University of Illinois at Urbana-Champaign

Abstract

In recent years, deep learning models have been widely used and are behind major breakthroughs across many fields. Deep learning models are usually considered to be black boxes due to their large model structures and complicated hierarchical nonlinear transformations. As deep learning technology continues to develop, the understanding of deep learning models is raising concerns, such as the understanding of the training and prediction behaviors and the internal mechanism of models.

In this thesis, we study the model understanding problem of deep neural networks from the perspective of piecewise linear property. First, we introduce the piecewise linear property. Next, we review the role and progress of deep learning understanding from the perspective of the piecewise linear property. The piecewise linear property reveals that deep neural networks with piecewise linear activation functions can generally divide the input space into a number of small disjointed regions that correspond to a local linear function within each region. Next, we investigate two typical understanding problems, namely model interpretation, and model complexity. In particular, we provide a series of derivations and analyses of the piecewise linear property of deep neural networks with piecewise linear activation functions. We propose an approach for interpreting the predictions given by such models based on the piecewise linear property. Next, we propose a method to provide local interpretation to a black box deep model by mimicking a piecewise linear approximation from the deep model. Then, we study deep neural networks with curve activation functions with the aim of providing piecewise linear approximations for these networks that would let them benefit from the piecewise linear property. After proposing a piecewise linear approximation framework, we investigate model complexity and model interpretation using the approximation. The thesis concludes by discussing future directions for understanding deep neural networks from the perspective of the piecewise linear property.

Keywords: deep learning, deep neural network, piecewise linear property, model interpretation, model complexity, linear regions

To my family.

"Reading and learning is to build up their own ideas and knowledge with the help of other people's ideas and knowledge."

— Pushkin

Acknowledgements

I would like to first express my deepest gratitude to my senior supervisor, Dr. Jian Pei, for his continuous guidance, supervision, encouragement, and help throughout my Ph.D. study. He has taught me a lot, not only research skills but also life wisdom. His insightful vision and profound knowledge guide me on how to become an excellent researcher. His rigorous attitude towards research and science sets a great example for me and inspires me to pursue excellence. I believe that everything he has taught me throughout my Ph.D. study will benefit my whole life.

I am very thankful to my supervisor Dr. Oliver Schulte, my thesis examiner Dr. Mo Chen, my external examiner Dr. Jingrui He, and the chair Dr. Jiannan Wang. They help exam my thesis and provide very insightful comments and advice for my research study and Ph.D. thesis.

During my Ph.D. study, I was fortunate to have a research internship in Microsoft Research Asia for a year. I am grateful to my intern mentor Dr. Weiqing Liu and my manager Dr. Jiang Bian, for their selfless help and advice for my research work and other problems. I have learned so much from them.

Many thanks to my colleagues, friends, and the faculty and support staff at Simon Fraser University, for their help and support during my Ph.D. study. A particular acknowledgment goes to Xiao Meng, Juhua Hu, Yu Yang, Chuancong Gao, Xiangbo Mao, Zicun Cong, Zijin Zhao, Yajie (Jill) Zhou, Biao Qin, Xi Wang, Chirong Zhang, Xuan Luo. All the wonderful times we spent together will become unforgettable and precious memories in my life.

Last but not least, I want to express the depth of my gratitude to my parents and my brother, for their unconditional love, support, and tolerance. I cannot complete my Ph.D. study and pursue my dream without their endless support and encouragement.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep learning models are widely used and promote strong improvements in many applications. However, because of their large model structures and complex hierarchical nonlinear transformations, deep learning models are generally regarded as black boxes. Understanding deep learning models is becoming more and more important, particularly understanding the training and prediction behaviors and internal mechanisms.

A series of works explores how to understand deep neural networks starting from piecewise linear activation functions or based on the piecewise linear property. The *piecewise linear property* describes how a deep neural network with piecewise linear activation functions divides the input space into multiple disjoint linear regions and corresponds to a local linear function in each linear region. The finite number of linear regions divided by such deep neural networks and the computable local linear functions benefit our understanding of deep neural networks.

Taking the effectiveness of the piecewise linear property as the starting point, in this thesis, we study two typical deep learning understanding problems from the perspective of piecewise linear property. First, we study the local interpretation of deep neural networks. The local interpretation attempts to interpret and attribute the prediction results given by deep learning models. The study of interpretability has significant practical utility and is conducive to a wide application of deep learning models. Second, we study the model complexity problem, which investigates the measure of model complexity for deep neural networks with fixed parameters. That is, it reviews how complex the functions represented by deep neural networks are. The model complexity problem is a cornerstone to studying many important theoretical problems, including optimization, generalization, and more.

1.1 Motivation

Due to its excellent performance, deep learning has been widely used in many practical applications, such as computer vision [56], natural language processing [72], and computational finance [106]. At the same time, the lack of understanding of deep learning models is raising increasing concerns. Deep learning models are regarded as black boxes because their behaviors and internal mechanisms are not fully understood. Despite extensive research so far, we still cannot explain the behaviors and effectiveness of deep learning models. For example, we cannot explain why a deep learning model predicts a given instance to a specific result [107]. We are unable to say why a deep learning model is effective and can generalize well [96, 97]. We are also not sure how complex a deep learning model should be selected for a certain application task [88, 94].

Understanding deep learning models is important for both theoretical study and practical applications. On one hand, understanding deep learning models involves addressing a series of fundamental problems such as interpretability, learnability, complexity, and generalization. Solving these problems is necessary to help establish a theoretical foundation for deep learning and to develop improvements in deep learning. On the other hand, wider applications in practical scenarios call for deep learning models to be more trustworthy and easy to understand. In particular, some governments and lawmakers have imposed strict requirements on the privacy and interpretability rights of models and data.

In recent years, many works explore to understand deep learning models. In particular, a group of studies attempts to understand deep neural networks from the perspective of piecewise linear property [4, 91, 104]. Deep neural networks with piecewise linear activation functions (e.g., ReLU, Hard Tanh) divide the input space into a large number of disjoint local regions. In each region, the deep neural network corresponds to a linear function. We call this characteristic the *piecewise linear property*.

Considering the piecewise linear property assists with investigating the understanding of deep neural networks in several ways. For example, Arora *et al.* [4] investigate the expressive power of deep neural networks by showing that deep neural networks with ReLU activation can represent all piecewise linear functions. Montufar *et al.* [91], Hanin and Rolnick [51], and Raghu *et al.* [104] investigate the complexity and learnability of deep neural networks by studying the number of linear regions of piecewise linear neural networks. Meanwhile, Che *et al.* [19] suggest adopting the Gradient Boosting Tree approximation to interpret the prediction behaviors of deep neural networks, which is essentially a piecewise linear approximation of the function of a deep neural network. Ribeiro *et al.* [107], Simonyan *et*

al. [114], and Smilkov *et al.* [115] propose to interpret specific prediction behaviors of a deep neural network in the local region by using a local linear approximation of the network.

The piecewise linear property is a good starting point for exploring the understanding of deep learning models for several reasons. First, the application of piecewise linear activation functions is becoming more and more common. The practical success of the piecewise linear activation functions, especially the Rectified Linear Unit (ReLU) [43, 95], drives the exploration and understanding of piecewise linear neural networks. Second, there are a finite number of linear regions divided by a neural network with piecewise linear activation functions. . The number of linear regions sheds light on analyzing the complexity [91], sensitivity [97], and loss surface of the network. In addition, the neural network within each linear region corresponds to a simple linear function with strong regional stability and easy to interpret.

In this thesis, we propose a systematic overview of the piecewise linear property of deep neural networks and their applications. We investigate two important problems in understanding deep neural networks. First, we study the local interpretation problem, which attempts to interpret the predictions made by a deep neural network on an arbitrary input. Next, we study the effective model complexity problem, which measures the effective model complexity of a deep neural network with fixed parameters. We adopt the model complexity measure to analyze the optimization process.

1.2 Overview of Contributions

In this thesis, we study the problem of understanding deep neural networks from the perspective of piecewise linear property. We mainly focus on two model understanding problems: the model interpretation and the model complexity. In particular, we make the following contributions:

- **Interpreting Deep Neural Networks with Piecewise Linear Activation Functions.** Another approach explores more precise and efficient interpretations of deep neural networks by taking into consideration the hierarchical structures of deep neural networks. We propose an approach to provide interpretations for deep neural networks with piecewise linear activation functions by analyzing the network structures. First, we prove that a piecewise linear neural network is mathematically equivalent to a set of local linear models, each of which corresponds to a convex polytope in the input space and acts on the group of instances within this convex polytope. Next, we propose a method called OpenBox to provide interpretations to piecewise linear neural networks by computing the equivalent set of local linear models in closed

form. We interpret the prediction results of a given arbitrary input instance using the decision features provided by the corresponding local linear model. In this way, we demonstrate the consistency of our provided interpretation within each convex polytope.

- **Interpreting Deep Neural Networks via Oblique Model Tree.** One typical approach to exploring the interpretation of deep neural networks is to mimic a simpler, interpretable student model from the deep neural network, ignoring the complex hierarchical transformations of the deep model. We propose using the oblique model tree structure to mimic the given deep neural network for model interpretation. The oblique model tree is a simple, easily-interpretable model that has sufficient expressive capacity to mimic a deep neural network. That is, the oblique model tree can always represent the function previously learned by an over-complicated deep neural network. In this thesis, we design an algorithm to efficiently train an oblique model tree to the required approximation degree. We also provide interpretations for predictions made by the given deep model through the learned oblique model tree. We demonstrate that the oblique model tree has good interpretability due to its tree structure. An oblique model tree acts as a piecewise linear approximation for a given “black-box" model and makes the prediction behaviors easily traceable.

- **Piecewise Linear Approximation of Deep Curve Neural Networks for Model Complexity.** The above two studies both focus on the understanding (especially the model interpretability) of deep neural networks from the perspective of the piecewise linear property. However, a noticeable problem is that not all deep neural networks are piecewise linear functions. The above exploration of piecewise linear neural networks cannot be directly applied to deep models with smooth curve activation functions. Motivated by these unexplored problems, we propose to investigate the understanding of deep neural networks with smooth curve activation functions, such as the Sigmoid and Tanh functions. To do this, we propose using the linear approximation neural network, which is a piecewise linear approximation to deep neural networks with smooth curve activation functions. This makes the understanding of these network structures also benefit from the piecewise linear property. Thus, understanding these network structures can also benefit from the piecewise linear property. Next, we design an algorithm to efficiently learn the piecewise linear approximation to the required approximation degree. Based on the piecewise linear approximation, we investigate the model complexity and model interpretability of deep neural networks

with smooth curve activation functions. Specifically, we provide a complexity measure and use this measure to investigate the model training process.

Our first two works devote to investigate the model interpretation study, based on the piecewise linear property. The third work proposes to investigate the model complexity problem, which is another fundamental problem to understand deep neural networks, based on the piecewise linear property. Furthermore, our proposed piecewise linear approximation bridges the gap between piecewise linear neural networks and non-piecewise linear neural networks. The approaches we proposed in our first two works and other state-of-the-art approaches based on the piecewise linear property can be easily applied to provide interpretations to the non-piecewise linear neural networks.

Parts of this thesis have already been published as conference papers or will be published soon. In particular, the main results in Chapter 3 have been published in [22], and the main results in Chapter 5 have been published in [63]. Part of the contents of Chapter 2 will be released as a survey paper soon.

1.3 Thesis Organization

The rest of this thesis is organized as follows:

- In Chapter 2, we introduce the piecewise linear property of deep neural networks, which is the most important background knowledge underpinning our research. We review the most current studies related to our research and review several decision tree structures that are typical example models having the piecewise linear property. Then, we review the latest studies on two typical deep neural network understanding problems: model interpretation and model complexity.

- In Chapter 3, we investigate the local interpretation problem of deep neural networks with piecewise linear activation functions. We study the piecewise linear property of deep piecewise linear neural networks. Based on the piecewise linear property, we devise an exact and consistent interpretation approach for predictions made by such networks.

- In Chapter 4, we propose a model-agnostic interpretation approach. We design a simple, interpretable model to mimic the deep model and provide interpretations using this simple model. We propose the oblique model tree structure to perform a piecewise linear approximation of the deep model and adopt the tree structure to interpret the predictions made by the deep neural network.

- In Chapter 5, we focus on deep neural networks with smooth curve activation functions. We propose a piecewise linear approximation approach to deep neural networks with smooth curve activation functions. We adopt this piecewise linear approximation to provide interpretations and model complexity measures for these non-piecewise linear neural networks.
- In Chapter 6, we conclude the thesis and propose several future directions that are worth exploring to understand deep neural networks by taking into consideration the piecewise linear property.

Chapter 2

Piecewise Linear Property and Related Works

In this chapter, we provide a brief introduction to the piecewise linear property of deep neural networks, which is the most important background knowledge of our research. We mainly introduce the concepts and newest results of the piecewise linear property. Then we review state-of-the-art studies of three directions, which are mostly related to our research. That is, the tree-structured model, interpretation of deep learning models, and complexity of deep learning models, respectively.

2.1 Piecewise Linear Property of Deep Neural Networks

To understand deep neural networks from the perspective of piecewise linear property, we first need to introduce and discuss the piecewise linear property. In general, when the activation function of a deep neural network is piecewise linear, the function represented by the network is also piecewise linear. In this section, we first provide a formal definition of deep neural networks, then introduce the piecewise linear property of deep neural networks.

2.1.1 Deep Neural Networks

Let $\mathcal{N}$ represent a deep neural network, f denote the function represented by network $\mathcal{N}$, short for $f_{\mathcal{N}} : \mathbb{R}^d \to \mathbb{R}^c$. d is the dimensionality of input features, and c the dimensionality of output. Given an arbitrary input instance x, the deep neural network generates prediction result $f(x)$. In particular, in classification tasks, the prediction result $f(x) \in \mathbb{R}^c$ is a vector which usually representing the probability score of x belonging to each class label.

There are multiple types of deep neural network architectures, including the deep feed-forward neural network, deep convolutional neural network, deep recurrent neural network, and others. The deep feed-forward neural network is a typical network architecture, which is the basis of a series of complex network architectures.

A deep feed-forward neural network consists of a series of fully connected layers. Each layer contains an affine transformation and a nonlinear activation function. Given an input instance $x \in \mathbb{R}^d$, a feed-forward neural network f can be written in the form of

$$f(x) = h_o \circ h_L \circ h_{L-1} \circ \ldots \circ h_1(x) \tag{2.1}$$

where $f(x) \in \mathbb{R}^c$ is the output vector corresponding to c class labels, h_o is the output layer in the form of $h_o(z) = V_o z + b_o$ where V_o and b_o are the weight matrix and the bias vector of the output layer, respectively.[1] L is the number of hidden layers, h_i is i-th hidden layer in the form of

$$h_i(z) = \phi(V_i z + b_i), \quad i = 1, \ldots, L \tag{2.2}$$

where V_i and b_i are the weight matrix and the bias vector of the i-th hidden layer, respectively. $\phi(\cdot)$ represent the activation function.

Deep neural networks are always highly nonlinear. One of the main reasons for the nonlinearity of deep networks is the nonlinear activation functions in hidden layers. The commonly used activation functions can be divided into two groups according to their algebraic properties, that is, piecewise linear activation functions and curve activation functions.

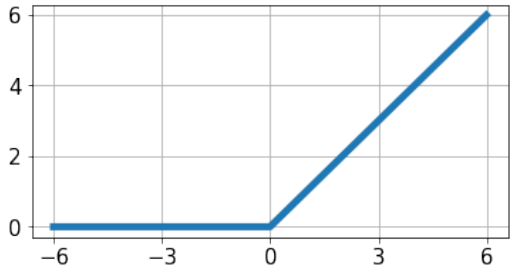
A **piecewise linear activation function** is composed of a finite number of pieces of affine functions. Some commonly used piecewise linear activation functions include ReLU [95], Leaky ReLU, hard Tanh [98], and others. ReLU activation function is short for Rectified Linear Unit and with the form of

$$\phi(z) = \begin{cases} 0 & \text{if} \quad z < 0 \\ z & \text{if} \quad z \geq 0 \end{cases} \tag{2.3}$$

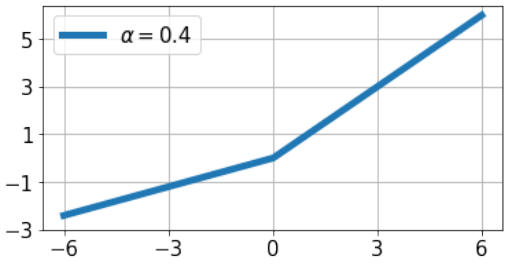
There are multiple variants of ReLU function, PReLU [55] is a typical one which is with the form of

$$\phi(z) = \begin{cases} az & \text{if} \quad z < 0 \\ z & \text{if} \quad z \geq 0 \end{cases} \tag{2.4}$$

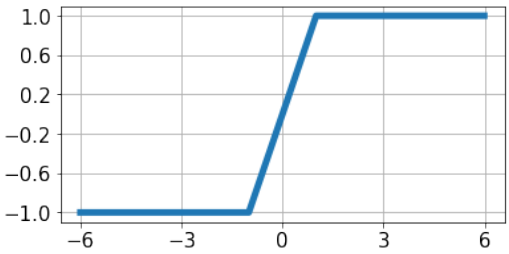
[1]Please note that we ignore the $softmax$ function here. The analysis of the piecewise linear property is based on the function before the $softmax$ function of the output layer.



(a) ReLU



(b) PReLU



(c) Hard Tanh

Figure 2.1: Three typical piecewise linear activation functions.

The hard Tanh activation function is with the form of

$$\phi(z) = \begin{cases} -1 & \text{if} \quad z < -1 \\ z & \text{if} \quad -1 \leq z \leq 1 \\ 1 & \text{if} \quad z \geq 1 \end{cases} \tag{2.5}$$

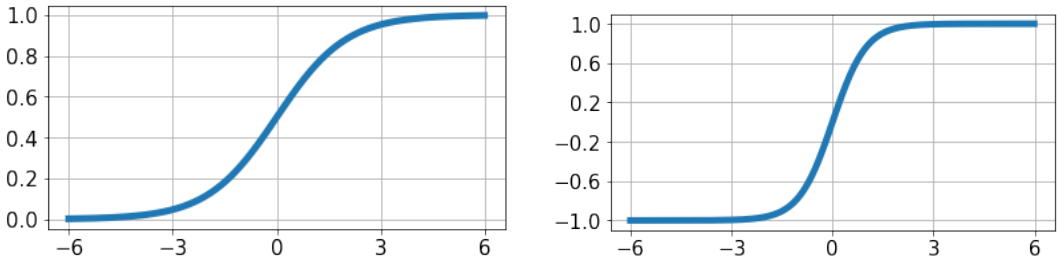
With piecewise linear activation functions, the deep neural network model f represents a continuous piecewise linear function.

A **curve activation function** is a continuous, differentiable nonlinear function whose geometric shape is a smooth curved line. Some commonly used curve activation functions include Sigmoid [43, 67], Tanh [66], and others. The Sigmoid activation function is with the form of

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

The Tanh activation function is with the form of

$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{2.7}$$



(a) Sigmoid (b) Tanh

Figure 2.2: Two typical smooth curve activation functions.

With smooth curve activation functions, the deep neural network model f represents a smooth curve function.

2.1.2 Piecewise Linear Property

A composition of multiple piecewise linear functions is still a piecewise linear function. The idea of piecewise linear function has been widely used in machine learning for decades of years, such as decision tree [11], adaptive logic network [3], etc. From this starting point, it is easy to conclude that given a deep neural network with piecewise linear activation functions (e.g., ReLU), the function represented by this deep neural network is piecewise linear [91, 104]. Each linear segment of such a piecewise linear neural network is called a linear region. The notion of "linear region" is introduced by Mantufar *et al.* [91] with the following formal definition:

Definition 1 (Linear region)**.** *A linear region of a piecewise linear function $f : \mathbb{R}^d \to \mathbb{R}^c$ is a maximal connected subset of the input space $\mathbb{R}^d$ on which f is linear.*

Raghu *et al.* [104] prove the following Theorem for linear regions generated by deep neural networks with piecewise linear activation functions.

Theorem 1. *Given the function of a neural network $\mathcal{N}$ with piecewise linear activation functions, the input space is partitioned into convex polytopes, with $\mathcal{N}$ corresponding to a different linear function on each linear region.*

Please see our Theorem 2 for proof of the convex polytope.

This theorem explains that a deep neural network with piecewise linear activation functions divides the input space into a series of linear regions [63, 91, 104], each of which is a convex polytope. Within each linear region, the deep neural network corresponds to a linear function. The property that a deep neural network with piecewise linear activation functions partitions the input space into linear regions is referred to as **piecewise linear property**.

The piecewise linear property is applicable to deep neural networks with any type of piecewise linear activation function. In other words, when the activation functions are piecewise linear, the neural network models have the piecewise linear property. Some widely used piecewise linear activation functions include ReLU [95], Leaky ReLU [84], PReLU [123], MaxOut [46], and others. Other piecewise linear activation functions such as the Heaviside function [65] can also keep the neural network piecewise linear property.

We propose the following conjecture of piecewise linear property:

Conjecture 1. *If the activation function ϕ is continuous piecewise linear, then the function of a neural network $\mathcal{N}$ with the continuous piecewise linear activation function ϕ is also continuous piecewise linear.*

This conjecture will not be discussed in detail in this thesis. We plan to investigate this in our future works.

The piecewise linear property provides novel opportunities for understanding deep learning models.

One typical deep learning understanding problem, model complexity, can be explored using the piecewise linear property. Model complexity explores how complex the function represented by a deep learning model is. It is a significant fundamental problem for understanding deep learning. A series of studies exploring the model complexity of deep neural networks start from piecewise linear activation functions or are based on the piecewise linear property [49, 83, 91, 104]. Montufar *et al.* [91] propose that the number of linear regions that have been generated by piecewise linear neural networks in the input space is a representation of model complexity. The authors then prove the theoretical upper and lower bounds of the maximal number of linear regions. Gühring *et al.* [49] connect piecewise linear neural networks to the family of functions in the Sobolev space, for the purpose of investigating the model complexity of piecewise linear neural networks. Raghu et al. [104] define the trajectory between two points in the input space and use the length of such a trajectory as a representation of model complexity. Then the authors prove the theoretical bounds to such a trajectory length, and empirically show the complexity measured by their proposed trajectory length. The piecewise linear property provides a good starting point for studying the model complexity of deep neural networks. We will review state-of-the-art studies of model complexity in detail in Section 2.2.3.

Another deep learning understanding problem, model robustness, can also be explored starting from the piecewise linear property. Model robustness explores the sensitivity of predictions to credible changes in input samples. Novak *et al.* [97] investigate the robustness of piecewise linear neural networks and propose two robustness estimations based on the

piecewise linear property. First, they suggest that when an instance is perturbed but still located within the same linear region in the input space, the robustness can be measured by the Jacobian norm. Second, they suggest that when an instance is perturbed to another linear region, the number of linear regions between the original region and the newly located linear region can be used to estimate model robustness. Their robustness measures make good use of the piecewise linear property of deep neural networks with piecewise linear activation functions.

The piecewise linear property also offers benefits for the study of model interpretation. Interpretation of deep neural networks is another important deep learning understanding problem that has raised significant concerns in recent years. Model interpretation explains the behaviors of a deep neural network, especially the prediction behavior when given an arbitrary input sample. One line of approach to model interpretation [107, 114, 115] is to propose a linear approximation to the deep model in a small region of the input space. The main idea is that when the region is small enough, it is able to use a linear function to approximate a nonlinear one to a close degree of approximation. Such a local linear approximation can be used to provide interpretations for the prediction behaviors in this small region. We will review several advanced studies on model interpretation in Section 2.2.2.

The piecewise linear property provides two major advantages for understanding deep neural networks. First, there are a finite number of linear regions partitioned by neural networks with piecewise linear activation functions. Although there is no closed-form method to calculate the number of linear regions, the number or density of linear regions can shed light on the properties of the loss surface [104] and how complex the function of the network is [63]. Second, within each linear region, the network corresponds to a linear function. Its simple form and linear characteristics make the linear function regionally stable and easy to interpret [22].

2.2 Related Works

In this section, we review several types of research works that are related to our work. In our work, we propose to understand deep neural networks from the perspective of piecewise linear property. We mainly focus on two types of understanding problems, the model interpretation of deep neural networks, and the model complexity of deep neural networks. Thus, in this section, we first review several tree structures. The decision tree structure is a typical kind of model that always has the piecewise linear property. Then, we review state-of-the-art studies on model interpretation of deep neural networks. Finally, we review state-of-the-art studies on the model complexity of deep neural networks.

2.2.1 Tree Structure

The decision tree is a typical type of machine learning models, which has been widely used and applied in many applications and problems. In general, the decision tree structure is defined by recursively dividing the input space, and building a local model in each resulting region of input space [92]. The typical decision tree structure for classification tasks uses axis-parallel hyperplanes for the input space division, and generates a certain class label for each resulting regions. In this case, decision tree structures generally have the piecewise linear property.

In this section, we introduce two decision tree structures that are related to our research. That is, the oblique decision tree [93] and the logistic model tree [73].

Oblique Decision Tree

Murthy *et al.* [93] proposed a tree structure named oblique decision tree in 1994. An oblique decision tree is a decision tree with an oblique split at each internal non-leaf node. Each inner node tests a linear combination of the features. The path from the root node to a leaf node forms the classification rule. Each leaf node represents a specific class label.

More formally, let $\mathbf{x} = \{x_1, x_2, \ldots, x_d\}$ denote the input sample with d features, an oblique split can be expressed as a linear combination of features, in the form of

$$a_0 + \sum_{i=1}^{d} a_i x_i \leq 0 \tag{2.8}$$

where $\{a_0, a_1, \ldots, a_d\}$ are the coefficients, x_i is the i-th feature of $\mathbf{x}$.

Comparing with the general decision trees which are axis-parallel, the oblique decision tree generates a splitting hyperplane at an oblique orientation to the axes, thus it is called the oblique decision tree. The oblique decision tree is obviously a more general form of axis-parallel trees. Murthy *et al.* [93] suggested that oblique decision trees produce polygonal divisions of the input space, while axis-parallel trees produce divisions in the form of hyper-rectangles that are parallel to the feature axes. In this case, when the underlying concept is defined by a polygonal space division, it is preferable to use oblique decision trees for classification. For example, there exist many domains in which one or two oblique hyperplanes will be the best model for classification.

Logistic Model Tree

Landwehr *et al.* [73] proposed the logistic model tree in 2005. A logistic model tree is a classification tree model that combines the logistic regression models with the tree induc-

tion [103]. Each inner node of a logistic model tree is associated with one of the features, and each leaf node corresponds to a logistic regression function. The segmentation of tree nodes is based on the increase of information gain. The LogitBoost algorithm [73] is used to fit the logistic regression function on each node in the tree.

More formally, a logistic model tree is made up of a set of inner nodes N and a set of leaf nodes T. Let S denote the whole input space spanned by all features of the data. The logistic model tree provides a disjoint partition of the input space S into regions S_t, written as

$$S = \bigcup_{t \in T} S_t, \quad S_t \cap S_{t'} = \emptyset \text{ for } t \neq t' \tag{2.9}$$

where each region S_t is represented by a leaf node in the tree, and has an associated logistic regression function. Let $x \in \mathbb{R}^d$ be an arbitrary input sample, d be the feature dimension of input data, let C be the number of class labels and C_j the j-th class label. The LogitBoost algorithm [30, 39] is proposed to build leaf-wise logistic regression functions by least-squares fits for each class label C_j, with the form of

$$L_j(x) = \beta_0 + \sum_{i=1}^{d} \beta_i x_i \tag{2.10}$$

where $\beta = \{\beta_0, \beta_1, \ldots, \beta_d\}$ denote the coefficient parameters and x_i the i-th value of x. Then the posterior probability can be computed by linear logistic regression in the form of

$$p(C_j|x) = \frac{e^{L_j(x)}}{\sum_{i=1}^{C} e^{L_i(x)}} \tag{2.11}$$

where $\sum_{j=1}^{C} L_j(x) = 1$.

Due to the good interpretability of tree structures, decision trees are widely used for interpreting deep neural networks at the early stage of model interpretation study. For instance, Zhang *et al.* [126] propose to learn a decision tree to explain predictions made by CNNs. The decision tree is used to decompose feature representations in Conv layers and provide semantic level interpretations to predictions. Schaaf *et al.* [109] introduce L^1-orthogonal regularization. Using the L^1-orthogonal regularization during training a deep model preserves the model accuracy and makes the model maintain low complexity. The authors show this regularization enhances the decision tree-based interpretations of deep neural networks. Liu *et al.* [81] propose a tree structure, called Linear Model U-trees, to interpret deep reinforcement learning models. They introduce a Linear Model U-tree based mimic learning framework for interpreting the Reinforcement Learning Environment.

2.2.2 Model Interpretation of Deep Neural Networks

The interpretability of deep learning is a widely concerned problem in recent years. A large number of works study the interpretability problem from multiple perspectives. A series of survey papers [16, 17, 50, 89, 127] summarizes these studies on the interpretability of deep learning models, We refer you to these survey works, such as Chakraborty *et al.* [17], Guidotti *et al.* [50], Carvalho *et al.* [16], for comprehensive and detailed studies of the interpretability of deep learning.

In this section, we briefly review state-of-the-art researches on the model interpretability of deep neural networks, which are related to our research. In particular, we review the interpretability studies on deep models that are well-trained already. The interpretability of pre-trained deep neural networks is a challenging problem and has attracted widespread attention in recent years.

Hidden Neuron Analysis

The hidden neuron analysis methods [31, 86, 124] interpret a pre-trained deep neural network by visualizing, revert-mapping, or labeling the features that are learned by the hidden neurons.

Yosinski *et al.* [124] visualized the live activations of the hidden neurons of a ConvNet, and proposed a regularized optimization to produce a qualitatively better visualization. Erhan *et al.* [34] proposed an activation maximization method and a unit sampling method to visualize the features learned by hidden neurons. Cao *et al.* [14] visualized a neural network's attention on its target objects by a feedback loop that infers the activation status of the hidden neurons. Li *et al.* [78] visualized the compositionality of clauses by analyzing the outputs of hidden neurons in a neural model for Natural Language Processing.

To understand the features learned by the hidden neurons, Mahendran *et al.* [86] proposed a general framework that revert-maps the features learned from an image to reconstruct the image. Dosovitskiy *et al.* [31] performed the same task as Mahendran *et al.* [86] did by training an up-convolutional neural network. Zhou *et al.* [128] interpreted a CNN by labeling each hidden neuron with a best aligned human-understandable semantic concept. However, it is hard to get a golden dataset with accurate and complete labels of all human semantic concepts.

The hidden neuron analysis methods provide useful qualitative insights into the properties of each hidden neuron. However, qualitatively analyzing every neuron does not provide much actionable and quantitative interpretation about the overall mechanism of the entire neural network [40].

Mimic Learning

By imitating the classification function of a neural network, the mimicking methods [5, 7, 18, 60] build a transparent model that is easy to interpret and achieves a high classification accuracy.

Ba *et al.* [5] proposed a model compression method to train a shallow mimic network using the training instances labeled by one or more deep neural networks. Hinton *et al.* [60] proposed a distillation method that distills the knowledge of a large neural network by training a relatively smaller network to mimic the prediction probabilities of the original large network. To improve the interpretability of distilled knowledge, Frosst and Hinton [40] extended the distillation method [60] by training a soft decision tree to mimic the prediction probabilities of a deep neural network. Che *et al.* [18] proposed a mimic learning method to learn interpretable phenotype features. Wu *et al.* [121] proposed a tree regularization method that uses a binary decision tree to mimic and regularize the classification function of a deep time-series model. Zhu *et al.* [130] built a forest model on top of a deep feature embedding network, however it is still difficult to interpret the deep feature embedding network.

The mimic models built by model mimicking methods are much simpler to interpret than deep neural networks. However, due to the reduced model complexity of a mimic model, there is no guarantee that a deep neural network with high expressive power and over-parameterization can be successfully imitated by a simpler shallow model. Thus, there is always a gap between the interpretation of a mimic model and the actual overall mechanism of the target deep neural network.

Local Interpretation

The local interpretation methods [36, 111, 115, 117] compute and visualize the important features for an input instance by analyzing the predictions of its local perturbations.

Simonyan *et al.* [114] generated a class-representative image and a class-saliency map for each class of images by computing the gradient of the class score with respect to an input image. Ribeiro *et al.* [107] proposed LIME to interpret the predictions of any classifier by learning an interpretable model in the local region around the input instance. Zhou *et al.* [129] proposed CAM to identify discriminative image regions for each class of images using the global average pooling in CNNs. Selvaraju *et al.* [110] generalized CAM [129] by Grad-CAM, which identifies important regions of an image by flowing class-specific gradients into the final convolutional layer of a CNN. Koh *et al.* [69] used influence functions to trace a model's prediction and identify the training instances that are the most responsible for the prediction.

The local interpretation methods generate an insightful individual interpretation for each input instance. However, the interpretations for perspectively indistinguishable instances may not be consistent [42] and can be manipulated by a simple transformation of the input instance without affecting the prediction result [68].

2.2.3 Model Complexity of Deep Neural Networks

Model complexity is a fundamental and theoretical problem that benefits the explorations of a lot of other problems, such as the generalization, and the model selection. The model complexity of classical machine learning models (e.g., decision tree, logistic regression) has been sufficiently explored and studies [12, 13, 79, 116] during the past decades years. However, the model complexity of deep neural networks is still in the exploratory stage so far.

In this section, we review the state-of-the-art studies on the model complexity of deep neural networks. We roughly summarize the model complexity studies into two categories, that is, the expressive capacity and the effective model complexity. We review the model complexity studies from these two categories.

Expressive Capacity

The expressive capacity of deep neural networks, also known as the expressive power of deep neural networks [8, 61, 104], devotes to study how complex functions can be represented by deep neural networks with certain model structures and certain sizes. In particular, how the model structure factors (i.e., layer width, network depth) affect expressive capacity is widely concerned.

The power of layer width of shallow neural networks is investigated [6, 26, 61, 85]. Hornik *et al.* [61] propose the universal approximation theorem, which proves that a single layer feedforward network with a finite number of neurons can approximate continuous functions under some mild assumptions. Later studies [6, 26, 85] advance this theorem. However, although with the universal approximation theorem, the layer width can be exponentially large.

Lu *et al.* [83] extend the universal approximation theorem to deep neural networks with bounded layer width. That is, a deep neural network that has bounded layer width and a finite number of layers (i.e., depth of the network) can still be a universal approximator. Recently, deep models are empirically discovered to be more effective than shallow ones. Subsequently, a series of studies focus on exploring the advantages of deep architecture in a theoretical view, which is called depth efficiency [8, 23, 33, 102]. Those studies show that the complexity of the function of a deep network can only be matched by a shallow network with exponentially more nodes. In other words, the function of deep architecture achieves

exponential complexity from layer to layer while incurs only polynomial complexity through layer width.

Some studies bound the complexity of models with respect to certain structures or activation functions [9, 27, 32, 91, 102]. Delalleau *et al.* [27] study the sum-product networks and use the number of monomials to reflect model complexity. Pascanu *et al.* [100] and Montufar *et al.* [91] investigate fully connected neural networks with piecewise linear activation functions (e.g. ReLU, Maxout), and use the number of linear regions generated by these models in the input space as a representation of model complexity. They theoretically bound the number of linear regions that a deep neural network with piecewise linear activation functions can generate. However, the study on model complexity in the view of the structure is not able to distinguish differences between two models with similar structures, which is needed for problems such as understanding the model training process.

Effective Complexity

The effective complexity of deep neural networks studies the effective, practical complexity of networks. That is, how nonlinear, how complex the function represented by a deep neural network is. Especially, the effective model complexity considers the values of model parameters and thus is expected to be sensitive to the different values of parameters.

Raghu *et al.* [104] propose a complexity measure for DNNs with piecewise linear activation functions. They follow previous studies on DNNs with piecewise linear activation functions and use the number of linear regions as a reflection of model complexity [91, 100]. To measure how many linear regions a data manifold is split, Raghu *et al.* [104] build a trajectory path from one input instance to another, then measure the number of linear region transitions through the trajectory path as an estimation of model complexity. Their trajectory length measure not only reflects the influences of model structures on model complexity, but also is sensitive to model parameters. They further study Batch Norm [64] using the complexity measure. Later, Novak *et al.* [97] generalize the trajectory measure to investigate the relationship between complexity and generalization of DNNs with piecewise linear activation functions.

However, the complexity measure using trajectory [104] cannot be directly generalized to smooth curve activation functions, such as Sigmoid, Tanh. In order to bridge this gap and investigate the effective complexity of models with smooth curve activation functions, in Chapter 4, we propose a complexity measure to deep neural networks with smooth curve activation functions. We propose a piecewise linear approximation to such neural networks. Following the idea of [91, 97, 100, 104], we use the number of linear regions of the piecewise linear approximation model as a representation of model complexity. There are two reasons.

First, since the number of linear regions reflects the nonlinearity, or complexity of a piecewise linear model [91, 104]. Similarly, the number of linear regions used to approximate a smooth curve function is a reflection of the complexity of the smooth curve function. Second, linear regions are finite in number and are detectable with the status of piecewise linear activation functions. This makes it a useful metric.

Chapter 3

Interpreting Deep Neural Networks with Piecewise Linear Activation Functions

Strong intelligent machines powered by deep neural networks are increasingly deployed as black boxes to make decisions in risk-sensitive domains, such as finance and medical. To reduce potential risk and build trust with users, it is critical to interpreting how such machines make their decisions. Existing works interpret a pretrained neural network by analyzing hidden neurons, mimicking pre-trained models, or approximating local predictions. However, these methods do not provide a guarantee of the exactness and consistency of their interpretations. In this chapter, we propose an elegant closed-form solution named *OpenBox* to compute exact and consistent interpretations for the family of Piecewise Linear Neural Networks. The major idea is to first transform a piecewise linear neural network into a mathematically equivalent set of linear classifiers, then interpret each linear classifier by the features that dominate its prediction. We further apply *OpenBox* to demonstrate the effectiveness of non-negative and sparse constraints on improving the interpretability world data sets demonstrate the exactness of consistency of our interpretation.

3.1 Introduction

More and more machine learning systems are making significant decisions routinely in important domains, such as medical practice, autonomous driving, criminal justice, and military decision making [45]. As the impact of machine-made decisions increases, the demand on clear interpretations of machine learning systems is growing ever stronger against the blind deployments of decision machines [47]. Accurately and reliably interpreting a machine learning model is the key to many significant tasks, such as identifying failure models [1],

building trust with human users [107], discovering new knowledge [105], and avoiding unfairness issues [125].

The interpretation problem of machine learning models has been studied for decades. Conventional models, such as Logistic Regression and Support Vector Machine, have all been well interpreted from both practical and theoretical perspectives [10]. Powerful non-negative and sparse constraints are also developed to enhance the interpretability of conventional models by sparse feature selection [62, 77]. However, due to the complex network structure of a deep neural network, the interpretation problem of modern deep models is yet a challenging field that awaits further exploration.

As reviewed in Chapter 2, the existing studies interpret a deep neural network in three major ways. The hidden neuron analysis methods [31, 86, 124] analyze and visualize the features learned by the hidden neurons of a neural network; the model mimicking methods [5, 7, 18, 60] build a transparent model to imitate the classification function of a deep neural network; the local explanation methods [36, 111, 115, 117] study the predictions on local perturbations of an input instance, so as to provide decision features for interpretation. All these methods gain useful insights into the mechanism of deep models. However, there is no guarantee that what they compute as an interpretation is truthfully the exact behavior of a deep neural network. As demonstrated by Ghorbani [42], most existing interpretation methods are inconsistent and fragile, because two perceptively indistinguishable instances with the same prediction result can be easily manipulated to have dramatically different interpretations.

Can we compute an exact and consistent interpretation for a trained deep neural network? In this chapter, we provide an affirmative answer, as well as an elegant closed form solution for the family of piecewise linear neural networks. Here, a **piecewise linear neural network** (**PLNN**) [52] is a neural network that adopts a piecewise linear activation function, such as MaxOut [46] and the family of ReLU [44, 55, 95]. The wide applications [74] and great practical successes [71] of PLNNs call for exact and consistent interpretations on the overall behaviour of this type of neural networks. We make the following technical contributions.

First, we prove that a PLNN is mathematically equivalent to a set of local linear classifiers, each of which being a linear classifier that classifies a group of instances within a convex polytope in the input space. Second, we propose a method named *OpenBox* to provide an exact interpretation of a PLNN by computing its equivalent set of local linear classifiers in closed form. Third, we interpret the classification result of each instance by the decision features of its local linear classifier. Since all instances in the same convex polytope share the same local linear classifier, our interpretations are consistent per convex polytope.

Fourth, we also apply *OpenBox* to study the effect of non-negative and sparse constraints on the interpretability of PLNNs. We find that a PLNN trained with these constraints selects meaningful features that dramatically improve the interpretability. Last, we conduct extensive experiments on both synthetic and real-world data sets to verify the effectiveness of our method.

The rest of this chapter is organized as follows. We formulate the problem in Section 4.2 and present *OpenBox* in Section 4.3. We report the experimental results in Section 4.4, and conclude the chapter in Section 4.5.

3.2 Problem Definition

A Piecewise Linear Neural Network, denoted by $\mathcal{N}$, is a neural network whose neurons in hidden layers adopt piecewise linear activation functions.

Given a piecewise linear neural network $\mathcal{N}$ with depth L, $\mathcal{N}$ contains $L-1$ hidden layers, we write the l-th hidden layer of $\mathcal{N}$ as h_l and the output layer as h_L. Let m_l represent the number of neurons in layer h_l; the total number of hidden neurons in network $\mathcal{N}$ is $m = \sum_{l=1}^{L-1} m_l$. Let $u_i^{(l)}$ denote the i-th neuron in h_l. Let $a^{(l)}$ denote the output of h_l, $z^{(l)}$ denote the weighted sum of the inputs to neurons in h_l. Let ϕ denote the activation function of hidden layers and $\phi_i^{(l)}$ specifically denote the activation function of i-th neuron in h_l. Let $W^{(l)}$ denote the weight matrix of h_l, $b^{(l)}$ the bias vector of h_l. For $l \in \{1, \ldots, L\}$, we compute $z^{(l)}$ by

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{3.1}$$

and compute $a^{(l)}$ for $l \in \{1, \ldots, L-1\}$ by

$$a^{(l)} = \phi(z^{(l)}) \tag{3.2}$$

An input instance of $\mathcal{N}$ is denoted by $x \in \mathcal{X}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ is a d-dimensional input space. The i-th dimension of x is denoted by x_i. The output of $\mathcal{N}$ is $a^{(L)} \in \mathcal{Y}$, where $\mathcal{Y} \subseteq \mathbb{R}^c$ is an c-dimensional output space. The activation function of the output layer h_L is the *softmax* function, written as

$$a^{(L)} = softmax(z^{(L)}) \tag{3.3}$$

Deep neural networks are black boxes. The function of a deep neural network with piecewise linear activation function is piecewise linear [91, 104]. However, due to the complicated structure and the huge number of linear segmentations, the piecewise linear neural network is not able to be directly interpreted and understood. How to interpret the overall

Notation	Description
$u_i^{(l)}$	The i-th neuron in layer l.
n_l	The number of neurons in layer l.
m	The total number of hidden neurons in $\mathcal{N}$.
$z_i^{(l)}$	The input of the i-th neuron in layer l.
$c_i^{(l)}$	The configuration of the i-th neuron in layer l.
$c(h)$	The h-th configuration of the piecewise linear neural network $\mathcal{N}$.
P_h	The h-th convex polytope determined by C_h.
$f_h(\cdot)$	The h-th linear classifier that is determined by C_h.
Q_h	The set of linear inequalities that define P_h.

Table 3.1: Frequently used notations.

mechanism of a piecewise linear neural network in an human understandable manner is an interesting problem that has attracted much attention in recent years.

Following a principled approach of interpreting a machine learning model [10], we regard an interpretation of a piecewise linear neural network $\mathcal{N}$ as the decision features that define the decision boundary of $\mathcal{N}$. We call a model interpretable if it explicitly provides its interpretation (i.e., decision features) in closed form.

Below we give the formal definition of the piecewise linear neural network interpretation problem.

Definition 2. *Given a piecewise linear neural network $\mathcal{N}$ with fixed structure and parameters, our task is to interpret the overall behavior of $\mathcal{N}$ by computing an interpretable model G that satisfies the following requirements.*

- ***Exactness.*** *G is mathematically equivalent to $\mathcal{N}$ such that the interpretations provided by G truthfully describe the exact behavior of $\mathcal{N}$.*
- ***Consistency.*** *G provides similar interpretations for classification of similar instances.*

Table 3.1 summarizes a list of frequently used notations.

Next, we introduce the *OpenBox* method that interprets the overall mechanism of a piecewise linear neural network in an exact and easy-to-understand manner.

3.3 OPENBOX Method

Mimicking a simple, interpretable model is an intuitive method of deep learning interpretation. However, the limitation of the efficiency and mimicking precision of mimic learning, especially when dealing with very huge deep learning models, prompts the proposal of more precise and efficient interpretation approaches. We study the piecewise linear property of deep neural networks with piecewise linear activation functions, then propose an exact and consistent interpretation approach to interpret the predictions made by deep models.

In this section, we describe the *OpenBox* method, which produces an exact and consistent interpretation of the given piecewise linear neural network by computing an interpretation model G in a piecewise linear closed form. We first define the configuration of a piecewise linear neural network $\mathcal{N}$, which specifies the activation status of each hidden neuron in $\mathcal{N}$. Then, we illustrate how to interpret the classification result of a fixed input instance. Last, we introduce how to interpret the overall behavior of $\mathcal{N}$ by computing an interpretation model G that is mathematically equivalent to $\mathcal{N}$.

3.3.1 The Configuration of a Piecewise Linear Neural Network

For a hidden neuron $u_i^{(l)}$, the piecewise linear activation function $\phi_i^{(l)}(z_i^{(l)})$ is in the following form.

$$\phi_i^{(l)}(z_i^{(l)}) = \begin{cases} r_1 z_i^{(l)} + t_1, & \text{if } z_i^{(l)} \in I_1 \\ r_2 z_i^{(l)} + t_2, & \text{if } z_i^{(l)} \in I_2 \\ \vdots & \\ r_k z_i^{(l)} + t_k, & \text{if } z_i^{(l)} \in I_k \end{cases} \tag{3.4}$$

where $k \geq 1$ is the number of linear segmentations of $\phi_i^{(l)}$, $\{r_1, \ldots, r_k\}$ are constant slopes, $\{t_1, \ldots, t_k\}$ are constant intercepts, and $\{I_1, \ldots, I_k\}$ is a collection of constant real intervals that partitions $\mathbb{R}$.

Given a fixed piecewise linear neural network $\mathcal{N}$, an input instance $x \in \mathcal{X}$ determines the value of $z_i^{(l)}$, and further determines which linear function in $\phi(z_i^{(l)})$ is applied. Based on which linear function is used, we encode the activation status of each hidden neuron by k states. Each state uniquely corresponds to one of the k linear functions of $\phi(z_i^{(l)})$. Let $c_i^{(l)}$ denote the state of $u_i^{(l)}$, we have $c_i^{(l)} \in \{1, \ldots, k\}$ and $z_i^{(l)} \in I_q$ if and only if $c_i^{(l)} = q$. Since the input to neurons are different from neuron to neuron, that is, $z_i^{(l)}$ might be different for different i and l, the states of different hidden neurons may differ from each other.

Based on the state of activation functions, we define the configuration as follow:

Definition 3. *Let $c^{(l)} = \{c_1^{(l)}, c_2^{(l)}, \ldots, c_{m_l}^{(l)}\}$ denote the states of all hidden neurons in h_l. We define the* ***configuration*** *of $\mathcal{N}$ as an m-dimensional vector which specifies the states of all hidden neurons in $\mathcal{N}$, written as*

$$c = \{c^{(1)}, \ldots, c^{(L-1)}\} \tag{3.5}$$

The configuration c of a piecewise linear neural network is uniquely determined by the input instance x. We write the function that maps an input instance $x \in \mathcal{X}$ to a configuration $c \in \{1, \ldots, k\}^N$ as $conf : \mathcal{X} \to \{1, \ldots, k\}^N$.

For a neuron $u_i^{(l)}$, denote by $r_i^{(l)}$ and $t_i^{(l)}$ the variables of slope and intercept of the linear function that corresponds to the state $c_i^{(l)}$. $r_i^{(l)}$ and $t_i^{(l)}$ are uniquely determined by $c_i^{(l)}$, such that $r_i^{(l)} = r_q$ and $t_i^{(l)} = t_q$ if and only if $c_i^{(l)} = q$.

For all hidden neurons in h_l, we write the variables of slopes and intercepts as $r^{(l)} = \{r_1^{(l)}, \ldots, r_{m_l}^{(l)}\}^\top$ and $t^{(l)} = \{t_1^{(l)}, \ldots, t_{m_l}^{(l)}\}^\top$, respectively. Then, we rewrite the activation function for all neurons in a hidden layer h_l as

$$\phi(z^{(l)}) = r^{(l)} \circ z^{(l)} + t^{(l)} \tag{3.6}$$

where $r^{(l)} \circ z^{(l)}$ is the Hadamard product between $r^{(l)}$ and $z^{(l)}$.

Next, we introduce how to interpret the classification result of a fixed input instance.

3.3.2 Exact Interpretation for the Prediction of a Fixed Instance

Given a piecewise linear neural network $\mathcal{N}$, we interpret the prediction made by $\mathcal{N}$ on a given input instance $x \in \mathcal{X}$ by deriving the closed form of $\mathcal{N}(x)$ using the configuration $c(x)$ as follows.

Following Equation 3.2 and Equation 3.6, we have that for all $l \in \{1, \ldots, L-1\}$

$$a^{(l)} = f(z^{(l)}) = r^{(l)} \circ z^{(l)} + t^{(l)} \tag{3.7}$$

By plugging $a^{(l)}$ into Equation 3.1, we rewrite $z^{(l+1)}$ as

$$\begin{aligned} z^{(l+1)} &= W^{(l+1)}(r^{(l)} \circ z^{(l)} + t^{(l)}) + b^{(l+1)} \\ &= (W^{(l+1)} \circ r^{(l)})z^{(l)} + W^{(l+1)}t^{(l)} + b^{(l+1)} \\ &= \tilde{W}^{(l+1)}z^{(l)} + \tilde{b}^{(l+1)} \end{aligned} \tag{3.8}$$

where $\tilde{b}^{(l+1)} = W^{(l+1)}t^{(l)} + b^{(l+1)}$, and $\tilde{W}^{(l+1)} = W^{(l+1)} \circ r^{(l)}$ is the extended version of Hadamard product, such that the entry at the i-th row and j-th column of $\tilde{W}^{(l+1)}$ is

$$\tilde{W}_{ij}^{(l+1)} = W_{ij}^{(l+1)} r_j^{(l)}$$

Then, by iteratively substituting Equation 3.8 into itself, we can write $z^{(l)}$ for all $l \in \{2, \ldots, L-1\}$ as

$$z^{(l)} = \prod_{q=0}^{l-2} \tilde{W}^{(l-q)} z^{(1)} + \sum_{i=2}^{l} \prod_{q=0}^{l-i-1} \tilde{W}^{(l-q)} \tilde{b}^{(i)}$$

By substituting $z^{(1)} = W^{(1)}x + b^{(1)}$ into the above equation, and let $\tilde{W}^{(1)} = W^{(1)}$, we rewrite $z^{(l)}$ for all $l \in \{1, \ldots, L\}$ as

$$\begin{aligned} z^{(l)} &= \prod_{q=0}^{l-1} \tilde{W}^{(l-q)} x + \sum_{i=1}^{l} \prod_{q=0}^{l-i-1} \tilde{W}^{(l-q)} \tilde{b}^{(i)} \\ &= \hat{W}^{(1:l)} x + \hat{b}^{(1:l)} \end{aligned} \tag{3.9}$$

where $\hat{W}^{(1:l)} = \prod_{q=0}^{l-1} \tilde{W}^{(l-q)}$ is the coefficient matrix of x, and $\hat{b}^{(1:l)}$ is the sum of the rest of the terms. The superscript $(1:l)$ indicates that $\hat{W}^{(1:l)}$ and $\hat{b}^{(1:l)}$ represent the overall computation of the given piecewise linear neural network on x from input layer to layer h_l.

The predicted probability given by $\mathcal{N}$ is the *softmax* of $z^{(L)}$, so the closed form of $\mathcal{N}(x)$ given input instance x is

$$\begin{aligned} \mathcal{N}(x) &= a^{(L)} \\ &= softmax(z^{(L)}) \\ &= softmax(\hat{W}^{(1:L)} x + \hat{b}^{(1:L)}) \end{aligned} \tag{3.10}$$

For a fixed piecewise linear neural network $\mathcal{N}$ and a fixed input instance x, the configuration $c = conf(x)$ is fixed. Therefore, $\hat{W}^{(1:L)}$ and $\hat{b}^{(1:L)}$ are constant parameters that are uniquely determined by the fixed c. $\mathcal{N}(x)$ reduces to a linear classifier in the form of Equation 3.10 whose decision boundaries are explicitly defined by $\hat{W}^{(1:L)}x + \hat{b}^{(1:L)}$.

Interpretation. Inspired by the interpretation method widely used by conventional linear classifiers, such as Logistic Regression and linear SVM [10], we interpret the prediction made by the network $\mathcal{N}$ on a given input instance x by the decision features of $\mathcal{N}(x)$. Specifically,

the entries of the i-th row of $\hat{W}^{(1:L)}$ reflects the feature importance of predicting x to the i-th class label.

Equation 3.10 provides a straightforward way to interpret the prediction result of a fixed input instance. However, individually interpreting the classification result of every single input instance is far from the understanding of the overall behavior of the piecewise linear neural network $\mathcal{N}$. Next, we describe how to interpret the overall behavior of the neural network by constructing an interpretation model G that is mathematically equivalent to $\mathcal{N}$.

3.3.3 Exact Interpretation of a Piecewise Linear Neural Network

A fixed piecewise linear neural network $\mathcal{N}$ with m hidden neurons has at most k^m configurations where k is the number of linear segmentations in the piecewise linear activation function. We represent the i-th configuration by $c(i) \in \mathcal{C}$, where $\mathcal{C} \subseteq \{1, \ldots, k\}^m$ is the set of all configurations of $\mathcal{N}$.

The volume of $\mathcal{C}$, denoted by $|\mathcal{C}|$, is bounded from upper by k^m. Each input instance $x \in \mathcal{X}$ has its corresponding configuration $conf(x) \in \mathcal{C}$. A configuration actually correspond to a linear region within the input space $\mathcal{X}$. Instances x_1, x_2 within the same linear region share the same configuration, written as $conf(x_1) = conf(x_2)$. Denote by $P_i = \{x \in \mathcal{X} \mid conf(x) = c(i)\}$ the set of input instances that have the same configuration $c(i)$. We prove in Theorem 2 that P_i is a convex polytope in $\mathcal{X}$ for any configuration $c(i) \in \mathcal{C}$.

Theorem 2. *Given a fixed piecewise linear neural network $\mathcal{N}$, for any configuration $c(i) \in \mathcal{C}$, $P_i = \{x \in \mathcal{X} \mid conf(x) = c(i)\}$ is a convex polytope in $\mathcal{X}$.*

Proof. We prove this theorem for neural networks with ReLU activation function. The result can be easily generalize to other piecewise linear activation functions. For ReLU, we prove this by proving that $conf(x) = c(i)$ is equivalent to a set of $2m$ linear inequalities with respect to x.

First, we prove that the input to each hidden layer, written as $z^{(l)}$ for any $l \in \{1, \ldots, L-1\}$, is a linear function of x. For $l = 1$, it follows Equation 3.1 that $z^{(1)} = W^{(1)}x + b^{(1)}$. For $l \in \{2, \ldots, L-1\}$, it follows Equation 3.9 that $z^{(l)} = \hat{W}^{(1:l)}x + \hat{b}^{(1:l)}$, which is a linear function of x, because $\hat{W}^{(1:l)}$ and $\hat{b}^{(1:l)}$ are constant parameters when $c(i)$ is fixed. In sum, $z^{(l)}$ is a linear function of x for all $l \in \{1, \ldots, L-1\}$.

Second, we prove that $conf(x) = c(i)$ is equivalent to a set of $2m$ linear inequalities with respect to x. Recall that $z_i^{(l)} \in C_q$ if and only if $c_i^{(l)} = q$. Denote by $\psi : \{1, \ldots, k\} \to \{I_1, \ldots, I_k\}$ the bijective function that maps a configuration $c_i^{(l)}$ to a real interval in $\{I_1, \ldots, I_k\}$, such that $\psi(c_i^{(l)}) = I_q$ if and only if $c_i^{(l)} = q$. Then, $conf(x) = c(q)$

Algorithm 1 *OpenBox*($\mathcal{N}, D_{train}$)

Require: a fixed piecewise linear neural network $\mathcal{N}$, the training set D_{train}.

Ensure: the set of active local linear classifiers G .

```
for each x ∈ D_train do
    Compute the configuration c(h) ← conf(x).
    if c(h) ∉ C then
        C ← C ∪ c(h).
        Compute the closed form of f_h(x) and P_h.
        G ← G ∪ (f_h(x), P_h).
    end if
end for
return G
```

is equivalent to a set of constraints, denoted by

$$Q_q = \{z_i^{(l)} \in \psi(c_i^{(l)}) \mid i \in \{1, \ldots, m_l\}, l \in \{1, \ldots, L-1\}\}. \tag{3.11}$$

Since $z_i^{(l)}$ is a linear function of x and $\psi(c_i^{(l)})$ is a real interval, each constraint $z_i^{(l)} \in \psi(c_i^{(l)})$ in Q_q is equivalent to two linear inequalities with respect to x.

Thus, $conf(x) = c(i)$ is equivalent to a set of $2m$ linear inequalities. As a result, P_i is a convex polytope in $\mathcal{X}$. □

According to the proof of Theorem 2, all input instances sharing the same configuration $c(i)$ form a unique convex polytope P_i that is explicitly defined by $2m$ linear inequalities in Q_i. $c(i)$ determines the linear classifier for the the convex polytope P_i, which is determined using Equation 3.10. All input instances in the convex polytope P_i are predicted using this linear classifier.

Denote by $f_i(x)$ the linear classifier of the polytope P_i, we write $\mathcal{N}(x)$ in the following piecewise linear form.

$$\mathcal{N}(x) = \begin{cases} f_1(x), & \text{if } x \in P_1 \\ f_2(x), & \text{if } x \in P_2 \\ \vdots & \\ f_m(x), & \text{if } x \in P_t \end{cases} \tag{3.12}$$

where $t = |\mathcal{C}|$ is the number of linear regions, $P_1 \cup \cdots \cup P_t = \mathcal{X}$ and $\forall i \neq j, P_i \cap P_j = \emptyset$.

According to Equation 3.12, we can interpret $\mathcal{N}$ as a set of **local linear classifiers**, each of which is a linear classifier $f_i(x)$ that classifies the input instances in convex polytope

region P_i. Denote by a tuple $(f_i(x), P_i)$ the i-th local linear classifier, the piecewise linear neural network $\mathcal{N}$ is mathematically equivalent to a set of local linear classifiers, denoted by

$$G = \{(f_i(x), P_i) \mid c(i) \in \mathcal{C}\}. \tag{3.13}$$

where G is our proposed interpretation model for $\mathcal{N}$.

Since a piecewise linear neural network with m hidden neurons and k-segment piecewise linear activation functions can have at most k^m configurations, G may contain at most k^m local linear classifiers. However, due to the hierarchical transformation of a deep learning model, the states of the hidden neurons in h_l highly depends on the states of the neurons in the former layers of h_l. Therefore, the volume of $\mathcal{C}$ might be much less than k^m. Thus, the number of local linear classifiers in G is much less than k^m.

In practice, we do not need to compute the entire set of local linear classifiers in G all at once. Instead, we can first compute an active subset of G, that is, the set of local linear classifiers that are actually used to classify the available set of input instances. Then, we can update G whenever a new local linear classifier is used to classify a newly coming input instance. We propose an *OpenBox* algorithm to compute G as the active set of local linear classifiers that are actually used to classify the set of training input instances, denoted by D_{train}. Algorithm 1 describes the *OpenBox* method.

The time cost of Algorithm 1 consists of the time T_{conf} to compute $conf(\mathbf{x})$ in step 3 and the time T_{lc} to compute the local linear classifier $(f_i(\mathbf{x}), P_i)$ in step 5. Since T_{conf} and T_{lc} are dominated by matrix multiplications, we evaluate the time cost of Algorithm 1 by the number of scalar multiplications. First, since we compute $conf(x)$ by forward propagating from layer h_1 to layer h_{L-1}, $T_{conf} = \sum_{l=1}^{L-1} n_l n_{l-1}$. Second, since $(f_i(\mathbf{x}), P_i)$ is determined by the set of tuples $\mathcal{G} = \{(\hat{W}^{(1:l)}, \hat{\mathbf{b}}^{(1:l)}) \mid l \in \{1, \ldots, L-1\}\}$, T_{lc} is the time to compute $\mathcal{G}$. Given $(\hat{W}^{(1:l-1)}, \hat{\mathbf{b}}^{(1:l-1)})$, we can compute $(\hat{W}^{(1:l)}, \hat{\mathbf{b}}^{(1:l)})$ by plugging $\mathbf{z}^{(l)}$ into Equation 3.8, and the time cost is $m_{l+1} m_l (m_l + 1)$. Since $\hat{W}^{(1:1)} = W^{(1)}$ and $\hat{\mathbf{b}}^{(1:1)} = \mathbf{b}^{(!)}$, we can iteratively compute $\mathcal{G}$. The overall time cost is $T_{lc} = \sum_{l=1}^{L-1} m_{l+1} m_l (m_l + 1)$. The worst case of Algorithm 1 happens when every input instance $x \in D_{train}$ has a unique configuration $conf(x)$. Let n denote the number of input instances in D_{train}, the time cost of Algorithm 1 in the worst case is $n(T_{conf} + T_{lc})$. Since $m_l, l \in \{1, \ldots, L-1\}$ are constant for a fixed neural network, the time cost of Algorithm 1 is $O(nd)$.

Interpretation. Now we are ready to introduce how to interpret the classification result of an input instance $x \in P_i, i \in \{1, \ldots, |\mathcal{C}|\}$. First, we interpret the classification result of x using the decision features of $f_i(x)$. Second, we interpret why x is contained in P_i using the polytope boundary features, which are the decision features of the polytope boundaries.

Models	PLNN	PLNN-NS	LR	LR-F	LR-NS	LR-NSF
NS	×	✓	×	×	✓	✓
Flip	×	×	×	✓	×	✓

Table 3.2: The models to interpret. LR is Logistic Regression. NS means non-negative and sparse constraints. Flip means the model is trained on the instances with flipped labels.

More specifically, a polytope boundary of P_i is defined by a linear inequality $z_q^{(l)} \in \psi(c_q^{(l)})$ in Q_i. By Equation 3.10, $z_q^{(l)}$ is a linear function with respect to x The polytope boundary features are the coefficients of x in $z_q^{(l)}$.

We also discover that some linear inequalities in Q_i are redundant whose hyperplanes do not intersect with P_i. To simplify our interpretation on the polytope boundaries, we remove such redundant-inequalities by Caron's method [15] and focus on studying the polytope boundary features of the non-redundant ones.

The advantages of *OpenBox* are three-fold as follows. First, our interpretation is exact, because the set of local linear classifiers in G are mathematically equivalent to the classification function of the piecewise linear neural network $\mathcal{N}$. Second, our interpretation is group-wise consistent. It is due to the reason that all input instances in the same convex polytope are classified by exactly the same local linear classifier, and thus the interpretations are consistent with respect to a given convex polytope. Last, our interpretation is easy to compute due to the low time complexity of Algorithm 1.

3.4 Experiments

In this section, we evaluate the performance of *OpenBox*, and compare it with the state-of-the-art method LIME [107]. In particular, we address the following questions:

(1) What do the local linear classifiers look like?

(2) Are the interpretations produced by LIME and *OpenBox* exact and consistent?

(3) Are the decision features of local linear classifiers easy to understand, and can we improve the interpretability of these features by non-negative and sparse constraints?

(4) How can one interpret the polytope boundary features of local linear classifiers?

(5) How effective are the interpretations of *OpenBox* in hacking and debugging a piecewise linear neural network model?

Data Sets	# Neurons $(m_1, m_2, \ldots, m_L)$	PLNN		PLNN-NS	
		$\lvert\mathcal{C}\rvert$	k^m	$\lvert\mathcal{C}\rvert$	k^m
SYN	$(2, 4, 16, 2, 2)$	266	2^{22}	41	2^{22}
FMNIST-1	$(784, 8, 2, 2)$	78	2^{10}	3	2^{10}
FMNIST-2	$(784, 8, 2, 2)$	23	2^{10}	18	2^{10}

Table 3.3: The network structures $(m_1, m_2, \ldots, m_L)$ and the number of configurations $|\mathcal{C}|$ of PLNN and PLNN-NS. The neurons in successive layers are initialized to be fully connected. $k = 2$ is the number of linear functions of ReLU, m is the number of hidden neurons.

Table 3.2 shows the details of the six models we used. PLNN represents piecewise linear neural network, PLNN-NS represents piecewise linear neural network with non-negative parameters (i.e., weights, bias). For both PLNN and PLNN-NS, we use the same network structure described in Table 3.3, and adopt the widely used activation function: ReLU [44]. We apply the non-negative and sparse constraints proposed by Chorowski *et al.* [21] to train PLNN-NS. Since our goal is to comprehensively study the interpretation effectiveness of *OpenBox* rather than achieving state-of-the-art classification performance, we use relatively simple network structures for PLNN and PLNN-NS, which are still powerful enough to achieve significantly better classification performance than Logistic Regression (LR). The decision features of LR, LR-F, LR-NS and LR-NSF are used as baselines to compare with the decision features of local linear classifiers.

The Python code of LIME is published by its authors[1]. The other methods and models are implemented in Matlab. PLNN and PLNN-NS are trained using the DeepLearnToolBox [99]. All experiments are conducted on a PC with a Core-i7-3370 CPU (3.40 GHz), 16GB main memory, and a 5,400 rpm hard drive running Windows 7 OS.

We use the following data sets. Detailed information of the data sets is shown in Table 3.4.

Synthetic (SYN) Data Set. As shown in Figure 3.2(a), this data set contains 20,000 instances uniformly sampled from a quadrangle in 2-dimensional Euclidean space. The red and blue points are positive and negative instances, respectively.

FMNIST-1 and FMNIST-2 Data Sets. Each of these data sets contains two classes of images in the Fashion MNIST dataset [122]. FMNIST-1 consists of the images of *Ankle Boot* and *Bag*. FMNIST-2 consists of the images of *Coat* and *Pullover*. Figure 3.1 show some

[1] `https://github.com/marcotcr/lime`

Data Sets	Training Data		Testing Data	
	# Positive	# Negative	# Positive	# Negative
SYN	6,961	13,039	N/A	N/A
FMNIST-1	4,000	4,000	3,000	3,000
FMNIST-2	4,000	4,000	3,000	3,000

Table 3.4: Detailed description of data sets.

example images in FMNIST-1 and FMNIST-2. All images in FMNIST-1 and FMNIST-2 are 28-by-28 grayscale images. We represent an image by cascading the 784 pixel values into a 784-dimensional feature vector. The Fashion MNIST data set is available online[2].

3.4.1 What Do the Local Linear Classifiers Look Like?

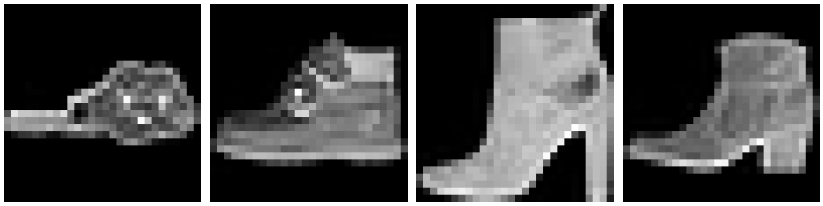
We demonstrate our claim in Theorem 2 by visualizing the local linear classifiers of the PLNN trained on SYN.

Figures 3.2(a)-(b) show the training instances of SYN and the prediction results of piecewise linear neural network on the training instances, respectively. Since all instances are used for training, the prediction accuracy is 99.9%.

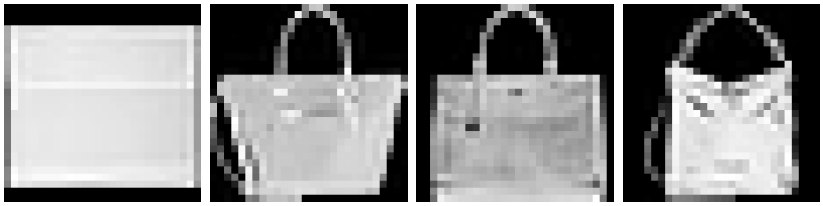
In Figure 3.2(c), we plot all instances with the same configuration in the same colour. Clearly, all instances with the same configuration are contained in the same convex polytope. This demonstrates our claim in Theorem 2.

Figure 3.2(d) shows the local linear classifiers whose convex polytopes cover the decision boundary of piecewise linear neural network and contain both positive and negative instances. As it is shown, the solid lines show the decision boundaries of the local linear classifiers, which capture the difference between positive and negative instances, and form the overall decision boundary of piecewise linear neural network. A convex polytope that does not cover the boundary of piecewise linear neural network contains a single class of instances. The local linear classifiers of these convex polytopes capture the common features of the corresponding class of instances. As to be analyzed in the following subsections, the set of local linear classifiers produce exactly the same prediction as piecewise linear neural network, and also capture meaningful decision features that are easy to understand.

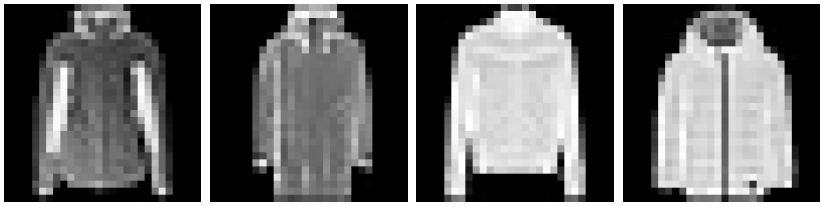
[2] `https://github.com/zalandoresearch/fashion-mnist`



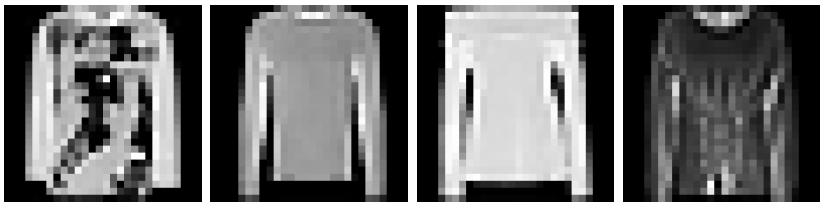
(a) *Ankle Boot* of FMNIST-1



(b) *Bag* of FMNIST-1

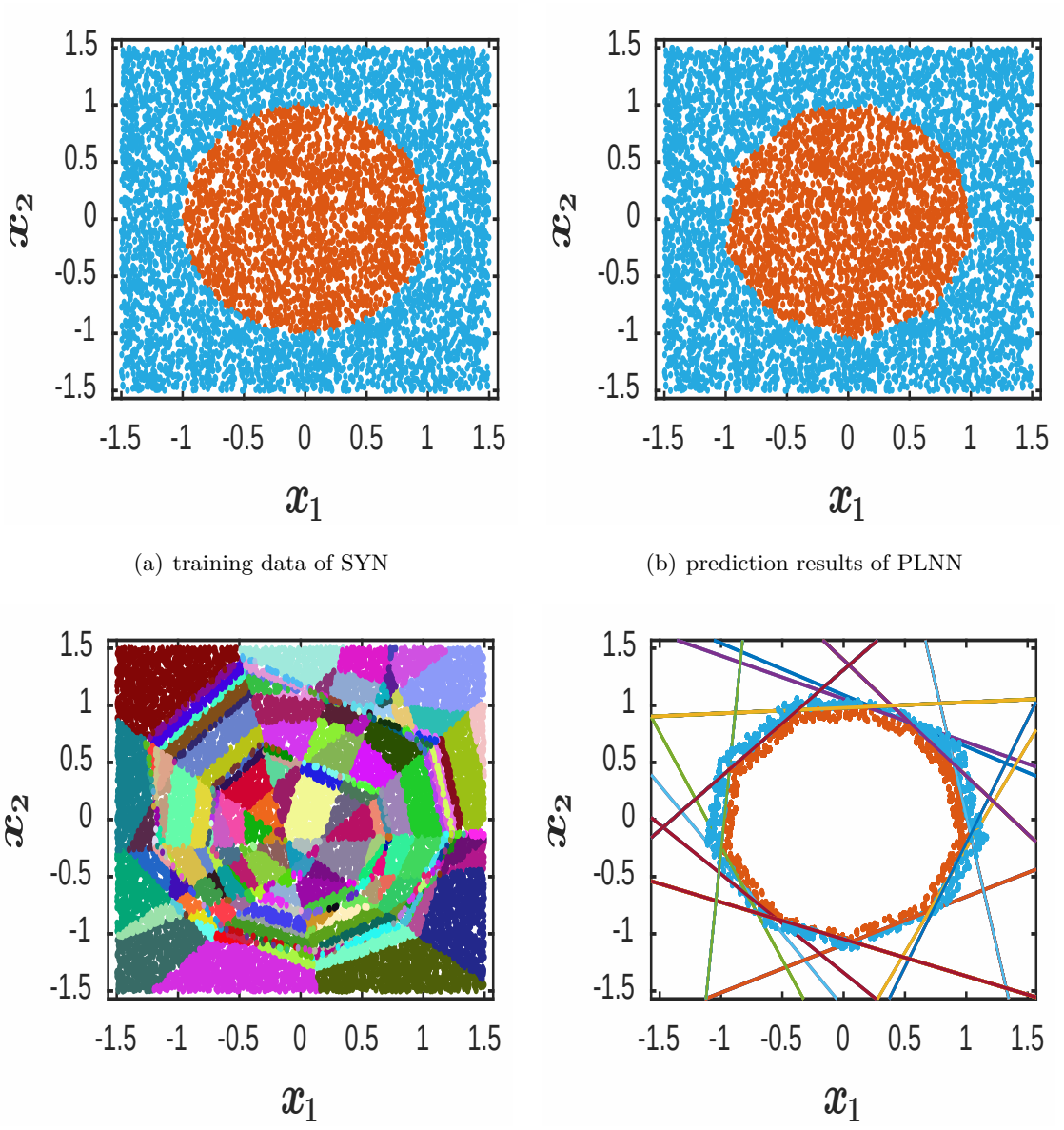


(c) *Coat* of FMNIST-2



(d) *Pullover* of FMNIST-2

Figure 3.1: Exempla images of FMNIST-1 and FMNIST-2.



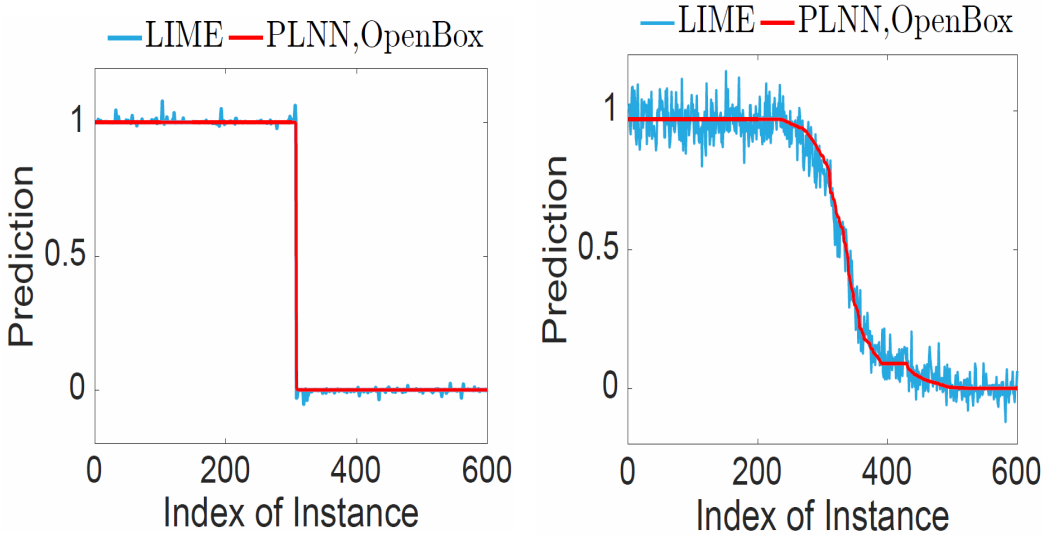
(a) training data of SYN

(b) prediction results of PLNN

(c) convex polytopes

(d) local linear classifiers

Figure 3.2: The local linear classifiers of the PLNN trained on SYN.



(a) FMNIST-1 (b) FMNIST-2

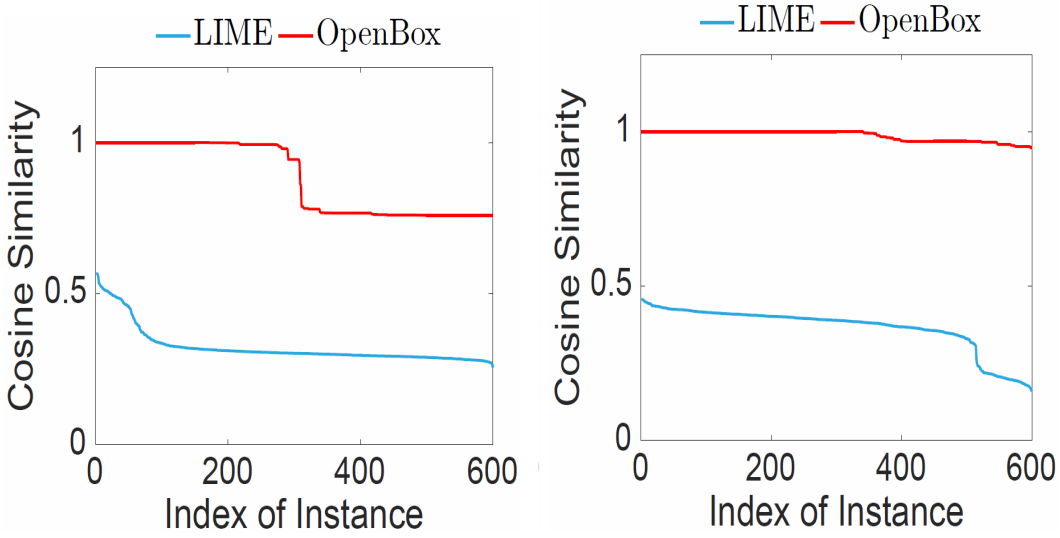
Figure 3.3: The predictions of LIME, OpenBox and PLNN. The predictions of all methods are computed individually. We sort the results by PLNN's predictions in descending order.

3.4.2 Are the Interpretations Exact and Consistent?

Exact and consistent interpretations are naturally favored by human minds. In this subsection, we systematically study the exactness and consistency of the interpretations of LIME and *OpenBox* on FMNIST-1 and FMNIST-2. Since LIME is too slow to process all instances in 24 hours, for each of FMNIST-1 and FMNIST-2, we uniformly sample 600 instances from the testing set, and conduct the following experiments on the sampled instances.

We first analyze the **exactness of interpretation** by comparing the predictions computed by the local interpretable model of LIME, the local linear classifiers of *OpenBox* and piecewise linear neural network, respectively. The prediction of an instance is the probability of classifying it as a positive instance.

In Figure 3.3, since LIME does not guarantee zero approximation error on the local predictions of piecewise linear neural network, the predictions of LIME are not exactly the same as piecewise linear neural network on FMNIST-1, and are dramatically different from piecewise linear neural network on FMNIST-2. The difference of predictions is more significant on FMNIST-2, because the images in FMNIST-2 are more difficult to distinguish, which makes the decision boundary of piecewise linear neural network more complicated and harder to approximate. We can also see that the predictions of LIME exceed $[0, 1]$. This is because the output of the interpretable model of LIME is not a probability at all. As a result, it is arguable that the interpretations computed by LIME may not truthfully describe



(a) FMNIST-1 (b) FMNIST-2

Figure 3.4: The cosine similarity (CS) between the decision features of each instance and its nearest neighbour. The results of LIME and *OpenBox* are separately sorted by cosine similarity in descending order.

the exact behavior of piecewise linear neural network. In contrast, since the set of local linear classifiers computed by *OpenBox* is mathematically equivalent to $F(\cdot)$ of piecewise linear neural network, the predictions of *OpenBox* are exactly the same as piecewise linear neural network on all instances. Therefore, the decision features of local linear classifiers exactly describe the overall behavior of piecewise linear neural network.

Next, we study the **interpretation consistency** of LIME and *OpenBox* by analyzing the similarity between the interpretations of similar instances.

In general, a consistent interpretation method should provide similar interpretations for similar instances. For an instance x, denote by x' the nearest neighbor of x by Euclidean distance, by $\gamma, \gamma' \in \mathbb{R}^d$ the decision features for the classification of x and x', respectively. We measure the consistency of interpretation by the cosine similarity between γ and γ', where a larger cosine similarity indicates a better interpretation consistency.

As shown in Figure 3.4, the cosine similarity of *OpenBox* is equal to 1 on about 50% of the instances, because *OpenBox* consistently gives the same interpretation for all instances in the same convex polytope. Since the nearest neighbours x and x' may not belong to the same convex polytope, the cosine similarity of *OpenBox* is not always equal to 1 on all instances. In constrast, since LIME computes individual interpretation based on the unique local perturbations of every single instance, the cosine similarity of LIME is signifi-

Data Set	FMNIST-1		FMNIST-2	
Accuracy	Train	Test	Train	Test
LR	0.998	0.997	0.847	0.839
LR-F	0.998	0.997	0.847	0.839
PLNN	**1.000**	**0.999**	**0.907**	**0.868**
LR-NS	0.772	0.776	0.711	0.698
LR-NSF	0.989	0.989	0.782	0.791
PLNN-NS	**1.000**	**0.999**	**0.894**	**0.867**

Table 3.5: The training and testing accuracy of all models.

cantly lower than *OpenBox* on all instances. This demonstrates the superior interpretation consistency of *OpenBox*.

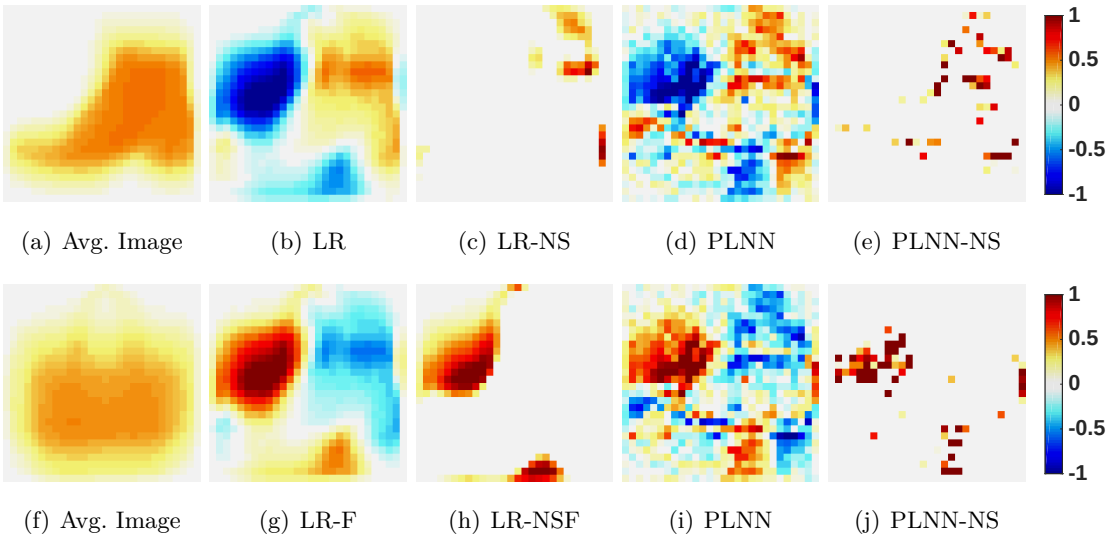
In summary, the interpretations of *OpenBox* are exact, and are much more consistent than the interpretations of LIME.

3.4.3 Decision Features of Local Linear Classifiers and the Effect of Non-negative and Sparse Constraints

Besides exactness and consistency, a good interpretation should also have a strong semantical meaning, such that the "thoughts" of an intelligent machine can be easily understood by a human brain. In this subsection, we first show the meaning of the decision features of local linear classifiers, then study the effect of the non-negative and sparse constraints in improving the interpretability of the decision features. The decision features of piecewise linear neural network and non-negative piecewise linear neural network are computed by *OpenBox*. The decision features of LR, LR-F, LR-NS and LR-NSF are used as baselines. Table 3.5 shows the accuracy of all models.

Figure 3.5 shows the decision features of all models on FMNIST-1. Interestingly, the decision features of piecewise linear neural network (PLNN) are as easy to understand as the decision features of LR and LR-F. All these features clearly highlight meaningful image parts, such as the ankle and heel of *Ankle Boot*, and the upper left corner of *Bag*. A closer look at the the average images suggests that these decision features describe the difference between *Ankle Boot* and *Bag*.

The decision features of piecewise linear neural network capture more detailed difference between *Ankle Boot* and *Bag* than the decision features of LR and LR-F. This is because the



(a) Avg. Image (b) LR (c) LR-NS (d) PLNN (e) PLNN-NS

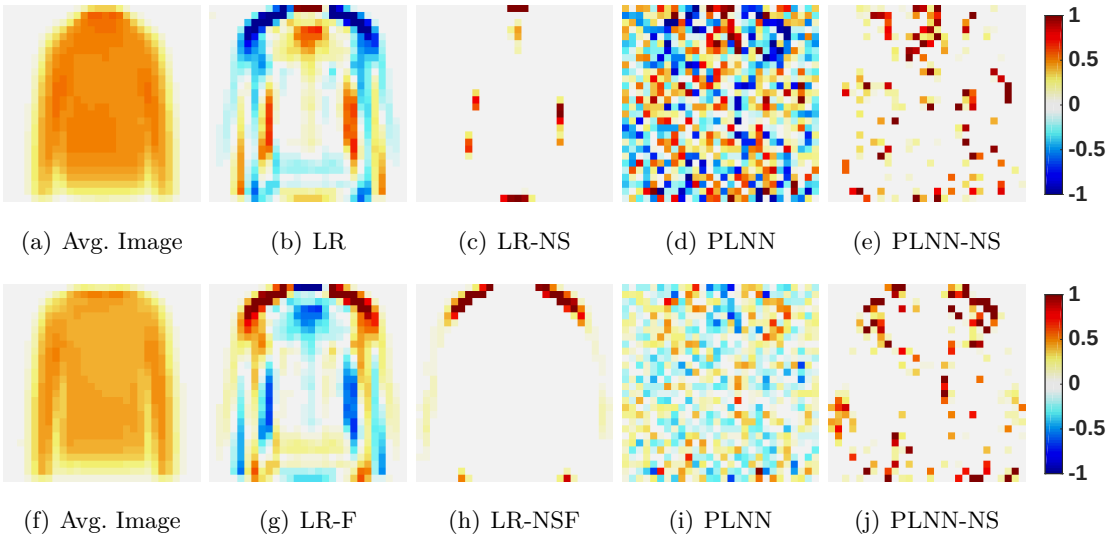
(f) Avg. Image (g) LR-F (h) LR-NSF (i) PLNN (j) PLNN-NS

Figure 3.5: The decision features of all models on FMNIST-1. (a)-(e) and (f)-(j) show the average image and the decision features of all models for *Ankle Boot* and *Bag*, respectively. For PLNN and PLNN-NS, we show the decision features of the local linear classifier whose convex polytope contains the most instances.

local linear classifiers of piecewise linear neural network only capture the difference between a subset of instances within a convex polytope, however, LR and LR-F capture the overall difference between all instances of *Ankle Boot* and *Bag*. The accuracies of piecewise linear neural network, LR and LR-F are comparable because the instances of *Ankle Boot* and *Bag* are easy to distinguish. However, as to be shown in Figure 3.6, when the instances are hard to distinguish, piecewise linear neural network captures much more detailed features than LR and LR-F, and achieves a significantly better accuracy.

Figure 3.6 shows the decision features of all models on FMNIST-2. As it is shown, LR and LR-F capture decision features with a strong semantical meaning, such as the collar and breast of *Coat*, and the shoulder of *Pullover*. However, these features are too general to accurately distinguish between *Coat* and *Pullover*. Therefore, LR and LR-F do not achieve a high accuracy. Interestingly, the decision features of piecewise linear neural network capture much more details than LR and LR-F, which leads to the superior accuracy of piecewise linear neural network.

The superior accuracy of piecewise linear neural network comes at the cost of cluttered decision features that may be hard to understand. Fortunately, applying non-negative and sparse constraints on piecewise linear neural network effectively improves the interpretability of the decision features without affecting the classification accuracy.



(a) Avg. Image (b) LR (c) LR-NS (d) PLNN (e) PLNN-NS

(f) Avg. Image (g) LR-F (h) LR-NSF (i) PLNN (j) PLNN-NS

Figure 3.6: The decision features of all models on FMNIST-2. (a)-(e) and (f)-(j) show the average image and the decision features of all models for *Coat* and *Pullover*, respectively. For PLNN and PLNN-NS, we show the decision features of the local linear classifier whose convex polytope contains the most instances.

In Figures 3.5 and 3.6, the decision features of PLNN-NS highlight similar image parts as LR-NS and LR-NSF, and are much easier to understand than the decision features of PLNN. In particular, in Figure 3.6, the decision features of PLNN-NS clearly highlight the collar and breast of *Coat*, and the shoulder of *Pullover*, which are much easier to understand than the cluttered features of piecewise linear neural network. These results demonstrate the effectiveness of non-negative and sparse constraints in selecting meaningful features. Moreover, the decision features of PLNN-NS capture more details than LR-NS and LR-NSF, thus PLNN-NS achieves a comparable accuracy with PLNN, and significantly outperforms LR-NS and LR-NSF on FMNIST-2.

In summary, the decision features of local linear classifiers are easy to understand, and the non-negative and sparse constraints are highly effective in improving the interpretability of the decision features of local linear classifiers.

3.4.4 Are Polytope Boundary Features of Local Linear Classifiers Easy to Understand?

The **polytope boundary features** (**PBFs**) of **polytope boundaries** (**PBs**) interpret why an instance is contained in the convex polytope of a local linear classifier. In this subsection, we study the semantical meaning of PBFs. Limited by space, we only use the

CP	$z_6^{(2)}$	$z_{11}^{(2)}$	$z_2^{(3)}$	$z_4^{(3)}$	*#Ankle Boot*	*#Bag*	Accuracy
1	/	> 0	> 0	/	3,991	3,997	0.999
2	≤ 0	> 0	/	≤ 0	9	0	1.000
3	/	≤ 0	/	> 0	0	3	1.000

Table 3.6: The PBs of the top-3 convex polytopes (CP) containing the most instances in FMNIST-1. "/" indicates a redundant linear inequality. Accuracy is the training accuracy of LLC on each CP.

CP	$z_4^{(2)}$	$z_5^{(2)}$	$z_8^{(2)}$	$z_2^{(3)}$	*#Coat*	*#Pullover*	Accuracy
1	> 0	> 0	> 0	> 0	3,932	3,942	0.894
2	> 0	≤ 0	> 0	> 0	32	10	0.905
3	> 0	≤ 0	≤ 0	> 0	18	0	0.944

Table 3.7: The PBs of the top-3 convex polytopes (CP) containing the most instances in FMNIST-2. Accuracy is the training accuracy of LLC on each CP.

PLNN-NS models trained on FMNIST-1 and FMINST-2 as the target model to interpret. The local linear classifiers of PLNN-NS are computed by *OpenBox*.

Recall that a PB is defined by a linear inequality $z_i^{(l)} \in \psi(c_i^{(l)})$, where the PBFs are the coefficients of x in $z_i^{(l)}$. Since the activation function is ReLU, $z_i^{(l)} \in \psi(c_i^{(l)})$ is either $z_i^{(l)} > 0$ or $z_i^{(l)} \leq 0$. Since the values of PBFs are non-negative for PLNN-NS, for a convex polytope P_h, if $z_i^{(l)} > 0$, then the images in P_h strongly correlate with the PBFs of $z_i^{(l)}$; if $z_i^{(l)} \leq 0$, then the images in P_h are not strongly correlated with the PBFs of $z_i^{(l)}$.

The above analysis of PBs and PBFs is demonstrated by the results in Tables 3.6 and 3.7, and Figure 3.7. Take the first convex polytope in Table 3.6 as an example, the PBs are $z_{11}^{(2)} > 0$ and $z_2^{(3)} > 0$, whose PBFs in Figures 3.7(b)-(c) show the features of *Ankle Boot* and *Bag*, respectively. Therefore, the convex polytope contains images of both *Ankle Boot* and *Bag*. A careful study of the other results suggests that the PBFs of the convex polytopes are easy to understand and accurately describe the images in each convex polytope.

We can also see that the PBFs in Figure 3.7 look similar to the decision features of PLNN-NS in Figures 3.5 and 3.6. This shows the strong correlation between the features learned by different neurons of PLNN-NS, which is probably caused by the hierarchy network structure. Due to the strong correlation between neurons, the number of configurations in $\mathcal{C}$ is much less than k^N, as shown in Table 3.3.

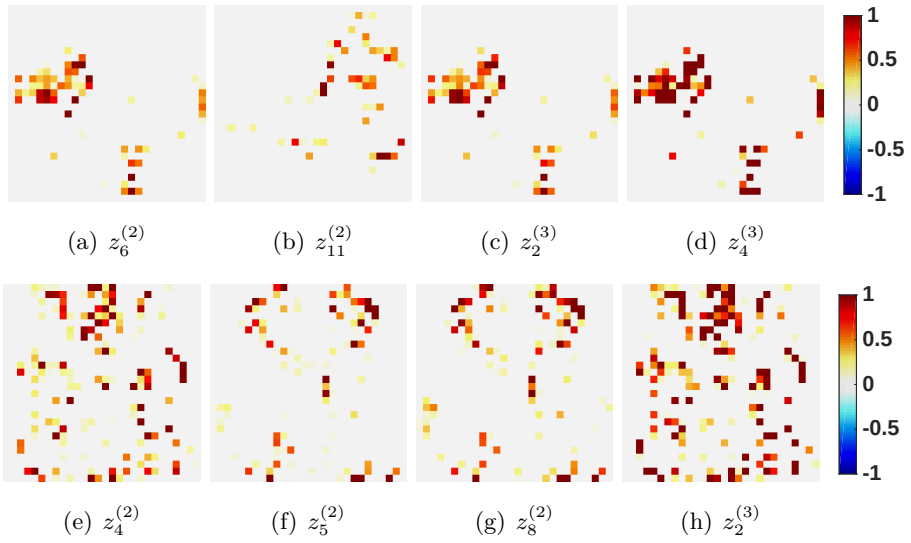


Figure 3.7: (a)-(d) show the PBFs of the PLNN-NS on FMNIST-1. (e)-(h) show the PBFs of the PLNN-NS on FMNIST-2.

Surprisingly, as shown in Table 3.7, the top-1 convex polytope on FMNIST-2 contains more than 98% of the training instances. On these instances, the training accuracy of local linear classifier is much higher than the training accuracies of LR-NS and LR-NSF. This means that the training instances in the top-1 convex polytope are much easier to be linearly separated than all training instances in FMNIST-2. From this perspective, the behavior of PLNN-NS is like a "divide and conquer" strategy, which set aside a small proportion of instances that hinder the classification accuracy such that the majority of the instances can be better separated by a local linear classifier. As shown by the top-2 and top-3 convex polytopes in Table 3.7, the set aside instances are grouped in their own convex polytopes, where the corresponding local linear classifiers also achieve a very high accuracy. Table 3.6 shows similar phenomenon on FMNIST-1. However, since the instances in FMNIST-1 are easy to be linearly separated, the training accuracy of PLNN-NS marginally outperforms LR-NS and LR-NSF.

3.4.5 Can We Hack a Model Using OpenBox?

Knowing what an intelligent machine "thinks" provides us the privilege to "hack" it. Here, to hack a target model is to significantly change its prediction on an instance $x \in \mathcal{X}$ by modifying as few features of x as possible. In general, the biggest change of prediction is achieved by modifying the most important decision features. A more precise interpretation of the target model reveals the important decision features more accurately, thus allows one to modify fewer features to achieve a bigger change of prediction. Following this idea, we apply

LIME and *OpenBox* to hack PLNN-NS, and compare the quality of their interpretations by comparing the change of PLNN-NS's prediction when modifying the same number of decision features.

For an instance $x \in \mathcal{X}$, denote by $\gamma \in \mathbb{R}^d$ the decision features for the classification of x. We hack PLNN-NS by setting the values of a few top-weighted decision features in x to zero, such that the prediction of PLNN-NS on x changes significantly. The change of prediction is evaluated by two measures as follows. First, the **change of prediction probability** (**CPP**) is the absolute change of the probability of classifying x as a positive instance. Second, the **number of label-changed instance** (**NLCI**) is the number of instances whose predicted label changes after being hacked. Again, due to the inefficiency of LIME, we use the sampled data sets in Section 3.4.2 for evaluation.

In Figure 3.8, the average CPP and NLCI of *OpenBox* are always higher than LIME on both data sets. This demonstrates that the interpretations computed by *OpenBox* are more effective than LIME when they are applied to hack the target model.

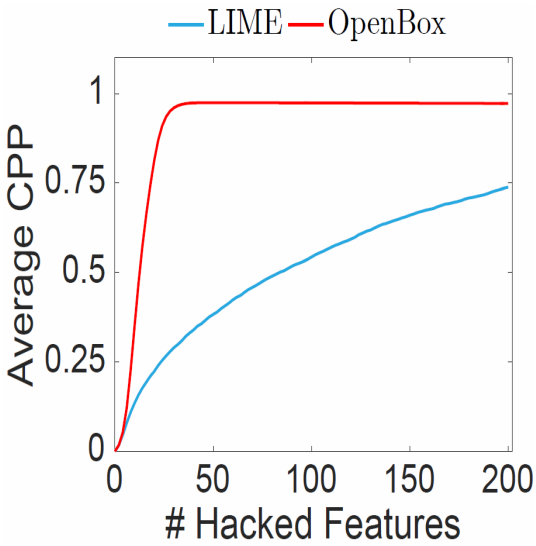
Interestingly, the advantage of *OpenBox* is more significant on FMNIST-1 than on FMNIST-2. This is because, as shown in Figure 3.3(a), the prediction probabilities of most instances in FMNIST-1 are either 1.0 or 0.0, which provides little gradient information for LIME to accurately approximate the classification function of the PLNN-NS. In this case, the decision features computed by LIME cannot describe the exact behavior of the target model.

In summary, since *OpenBox* produces the exact and consistent interpretations for a target model, it achieves an advanced hacking performance over LIME.

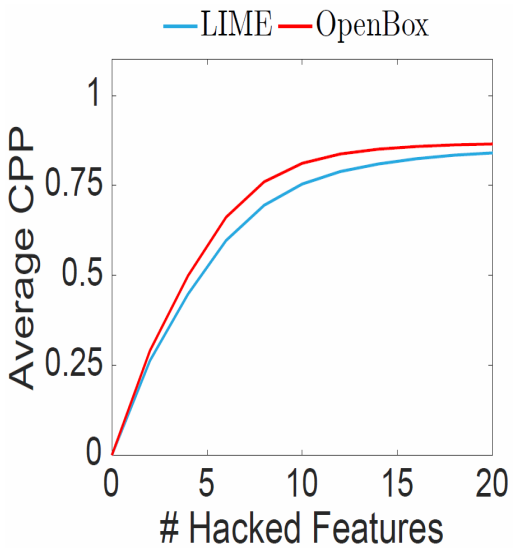
3.4.6 Can We Debug a Model Using OpenBox?

Intelligent machines are not perfect and predictions fail occasionally. When such failure occurs, we can apply *OpenBox* to interpret why an instance is mis-classified.

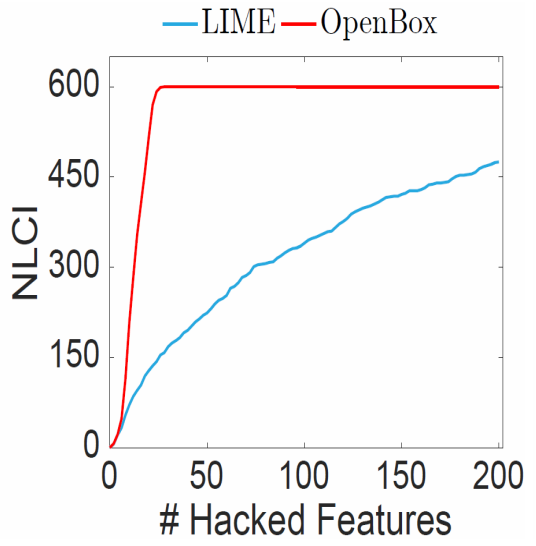
Figure 3.9 shows some images that are mis-classified by PLNN-NS with a high probability. In Figures 3.9(a)-(c), the original image is a *Coat*, however, since the scattered mosaic pattern on the cloth hits more features of *Pullover* than *Coat*, the original image is classified as a *Pullover* with a high probability. In Figures 3.9(d)-(f), the original image is a *Pullover*, however, it is mis-classified as a *Coat* because the white collar and breast hit the typical features of *Coat*, and the dark shoulder and sleeves miss the most significant features of *Pullover*. Similarly, the *Ankle Boot* in Figure 3.9(g) highlights more features on the upper left corner, thus it is mis-classified as a *Bag*. The *Bag* in Figure 3.9(j) is mis-classified as an *Ankle Boot* because it hits the features of ankle and heel of *Ankle Boot*, however, misses the typical features of *Bag* on the upper left corner.



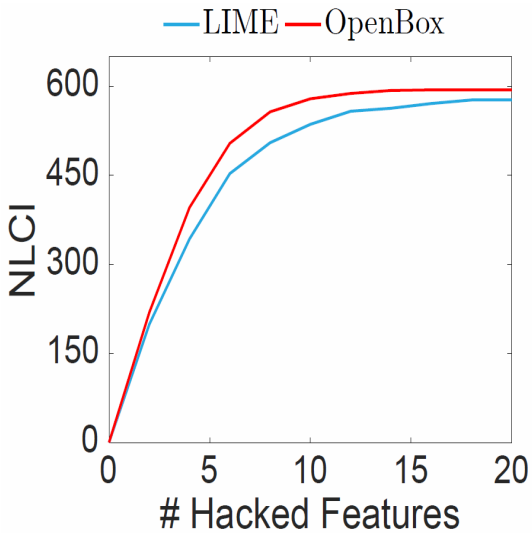
(a) FMNIST-1



(b) FMNIST-2



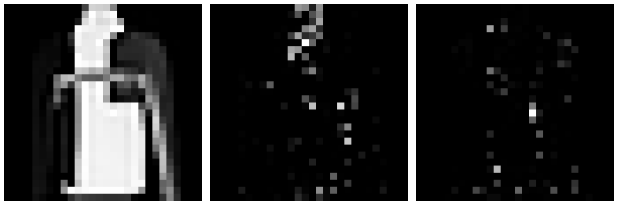
(c) FMNIST-1



(d) FMNIST-2

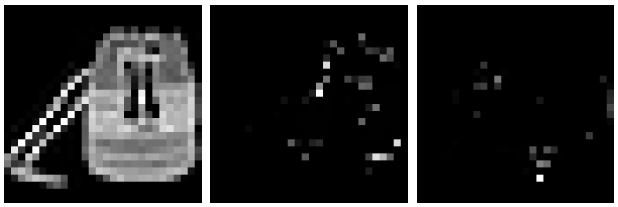
Figure 3.8: The hacking performance of LIME and *OpenBox*. (a)-(b) show the average (Avg.) CPP. (c)-(d) show the NLCI.

(a) CO (b) CO: 0.04 (c) PU: 0.96



(d) PU (e) CO: 1.00 (f) PU: 0.00

(g) AB (h) AB: 0.16 (i) BG: 0.84



(j) BG (k) AB: 1.00 (l) BG: 0.00

Figure 3.9: The mis-classified images of (a) *Coat* (**CO**), (d) *Pullover* (**PU**), (g) *Ankle Boot* (**AB**), and (j) *Bag* (**BG**). (a), (d), (g) and (j) show the original images. For the rest of the subfigures, the caption shows the prediction probability of the corresponding class; the image shows the decision features supporting the prediction of the corresponding class.

In conclusion, as demonstrated by Figure 3.9, *OpenBox* accurately interprets the misclassifications, which is potentially useful in debugging abnormal behaviors of the interpreted model.

3.5 Summary

In this chapter, we tackle the challenging problem of interpreting piecewise linear neural networks. By studying the states of hidden neurons and the configuration of a piecewise linear neural network, we prove that a piecewise linear neural network is mathematically equivalent to a set of local linear classifiers, which can be efficiently computed by the proposed *OpenBox* method. Extensive experiments show that the decision features and the polytope boundary features of local linear classifiers provide exact and consistent interpretations on the overall behavior of a piecewise linear neural network. Such interpretations are highly effective in hacking and debugging piecewise linear neural network models.

There are some possible future directions to follow up on this project. First, the application of the *OpenBox* framework can be extended. As we discussed in this chapter, this framework can efficiently provide a trustworthy interpretation of piecewise linear neural networks. However, can we extend this interpretation framework to interpret more general neural networks that adopt smooth curve activation functions, such as Sigmoid and Tanh? Second, we should consider the user experiences when giving interpretations. When designing to provide interpretations, the target users of the interpretation and their easy-to-understand evaluation of these interpretations are important factors. In this case, taking into consideration user experiences to evaluate the quality of interpretation approaches is necessary. The first question is solved in Section 5, where we propose a piecewise linear approximation to neural networks with smooth curve activation functions, then use *OpenBox* to give interpretations. We plan to investigate the second question in future works, to propose some possible solutions to bring user experiences into model interpretations.

Chapter 4

Interpreting Deep Neural Networks via Oblique Model Tree

In this chapter, we investigate the deep learning model interpretation problem. Recently, with rapid development and wide application of deep learning, the significance of the interpretability of deep learning models, especially in practical applications, attracts more and more concerns. We propose a model-agnostic interpretation approach, namely the oblique model tree (OMT), to provide interpretations to predictions made by deep learning models. The oblique model tree is a tree structure that provides interpretations to a deep neural network by mimicking from the deep neural network. We devise an algorithm to efficiently build the oblique model tree. Our experiments on one text dataset and two image datasets demonstrate the effectiveness of the oblique model tree. That is, the oblique model tree is able to closely mimic the original deep model and provide easy-understandable interpretations to the deep model.

4.1 Introduction

During recent years, deep neural networks (e.g., DNN, CNN) have been widely used and achieve great success in many areas, including image classification [71], recommendation system [24], speech recognition [48] and game events such as Go game [112]. At the same time, the lack of interpretability is realized to be a big challenge of deep neural networks, which limit the usage of deep neural networks in some practical applications.

Interpretability of deep neural networks is not only important but necessary, especially for some practical applications. One typical example is the usage of deep neural networks in healthcare. Suppose a deep neural network is used to provide diagnosis predictions, an interpretation of why and how a diagnosis is given should be provided to patients, together with a diagnosis prediction made by the model. Such an interpretation is necessary to let clinical staff adopt the prediction, and also let the patient trust it [19]. Another example

is self-driving. A piece of news in May 2018 tells that an Uber self-driving car killed a pedestrian. [1] This shocking news brings a discussion on the safety and trustworthiness of deep learning. To ensure the safety of a deep learning model and make a deep learning model trustworthy, interpretations of the deep learning model (i.e., interpretations of predictions and internal mechanisms) is necessary. Further, when accidents happen, such interpretations can help us find out how it happens and how to improve the model.

A series of works investigate the interpretability of deep neural networks from multiple perspectives [18, 31, 86, 111]. One line of works proposes to interpret deep neural networks that are already well-trained. These approaches are called post-hoc interpretation approaches [80].

There are two typical types of post-hoc interpretation approaches. The first typical type of post-hoc interpretation approaches is called global interpretation, which proposes to analyze and explain the whole logic of a deep neural network model. Analyzing the whole logic is to analyze the behaviors of the model, the functions represented by the model, and the effectiveness of hidden layers and neurons. Global interpretation can be given by visualizing layers and units of the deep model to provide meaningful labels ranging from colors, materials, textures, and scenes to the individual units [128]. Interpreting layers and units helps to understand the structure and behaviors of a deep model. The second typical post-hoc interpretation approach is called local interpretation, which provides interpretations to the prediction results given by a deep neural network on an arbitrary input sample. That is to provide explanations to how and why a prediction occurred (i.e., make a diagnosis). Local interpretation can be given by learning a sparse linear model in a local region near a particular instance [107], or by analyzing the gradient of class scores with respect to features of a particular instance [114].

These approaches focus on different views of interpretations. In this case, a major challenge is: how can we achieve local interpretation and global understanding at the same time? Local interpretation explains the prediction made by the deep model on a given input instance. Meanwhile, global understanding is expected to help explain the global mechanism of the whole deep model.

Mimic learning provides a solution. Mimic learning is also known as knowledge distillation [5]. Interpreting by mimic learning is to mimic a complicated deep model using an interpretable simple model such as decision tree [126] and gradient boosting tree[19]. One major challenge of these approaches is when the deep model is very complicated, the mim-

[1] https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian

icked model itself grows to be very complex and may be difficult to interpret. This motivates us to explore the possibility of proposing a new model structure to explore interpretation via mimic learning. Such a model structure is expected to achieve similar performance as the deep model and at the same time easy to interpret.

Motivated by this, we propose an oblique model tree structure, which offers interpretations to a given deep model by mimicking the deep models. The oblique model tree consists of an oblique decision tree with logistic regression classifier at leaf nodes. The oblique model tree is a simpler model with lower complexity than deep neural networks, and in general, has good interpretability along with strong learning capacity. We devise a tree building algorithm for oblique model trees, which is designed to take into consideration both the tree splitting process and the logistic regression classifiers of leaf nodes. We conduct extensive experiments on several deep learning architectures including deep feedforward neural network(DNN) and convolutional neural network(CNN) for binary classification tasks on both text and image datasets. The experimental results demonstrate that our oblique model tree architecture has a strong learning capacity of the deep model, and maintains the competitive prediction performance. Meanwhile, we prove that the interpretations given by an oblique model tree to the original deep model are clear and easy-understandable by sufficient qualitative and quantitative analysis.

The rest of this chapter is organized as follows. In Section 4.2, we introduce the framework of interpreting by mimic learning. In Section 4.3, we introduce the oblique model tree structure and devise an algorithm for building an oblique model tree. In Section 4.4, we report the experimental results. Finally, in Section 4.5 we present the conclusion of the work.

4.2 Interpretation by Mimic Learning

Deep neural networks are regarded as unexplainable black boxes due to their complicated nonlinear transformations through hierarchical structures. As deep neural networks are widely used in many applications, the interpretation problem of deep neural networks attracts more and more attention. The approach of treating the network as a 'black box' and providing interpretations to predictions made by this 'black box' model is called *model-agnostic interpretation* [17, 107]. One typical model-agnostic interpretation approach is to mimic a simple, interpretable model from the black-box network.

4.2.1 Mimic Learning

Mimic learning is also known as teacher-student model [19], knowledge distillation [60]. The main idea is, given a large, complex and high-accurate model, to transfer knowledge

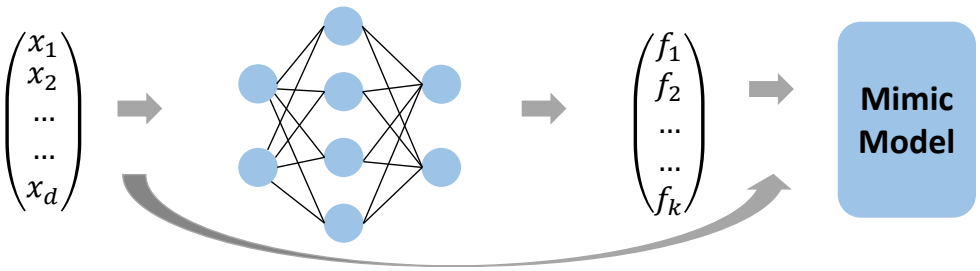


Figure 4.1: Pipeline of mimic learning.

from this model to a smaller, faster, yet still accurate model. The large model is the teacher model which is used to train the small student model. The method of mimicking the teacher model is to utilize the soft labels of the teacher model as target labels to train a student mimic model. The soft label is the prediction results given by the teacher model, such as the prediction probability value in classification tasks.

Figure 4.2 shows the training and prediction pipelines of the mimic learning framework. In particular, in the training pipeline, first, a deep model (e.g., DNN, CNN) is well trained based on training data and the corresponding class labels. For each input sample, the well-trained deep model generates a prediction result, that is, the soft label of the input sample. By feeding the input samples and corresponding soft labels, the student mimic model is trained to minimize the gap between the output of the student mimic model and that of the teacher deep model.

There are two reasons why the mimic learning approach works well on deep neural networks. First, deep neural networks are always over-parameterized [60]. In this case, the practical model complexity of a trained network might be much lower than the expressive power of the model. This provides room for compressing the network or distilling knowledge into a smaller neural network. Second, the teacher model can eliminate some potential noise and error in the training data, and its soft labels are usually more informative than the original labels, thus can make the learning of the student model easy [19].

4.2.2 Interpreting by Mimic Learning

The main idea of interpreting by mimic learning is to learn a simple, interpretable mimic model from the target model [5, 19, 40, 57, 121, 126]. The target model is the deep neural network we explore to interpret. The mimic model approximates the predictions made by the target network. When the prediction given by the mimic model is similar to that of the target network, the mimic model is the functional equivalent to the target model [107], and is able to explain the prediction behaviors of the target model.

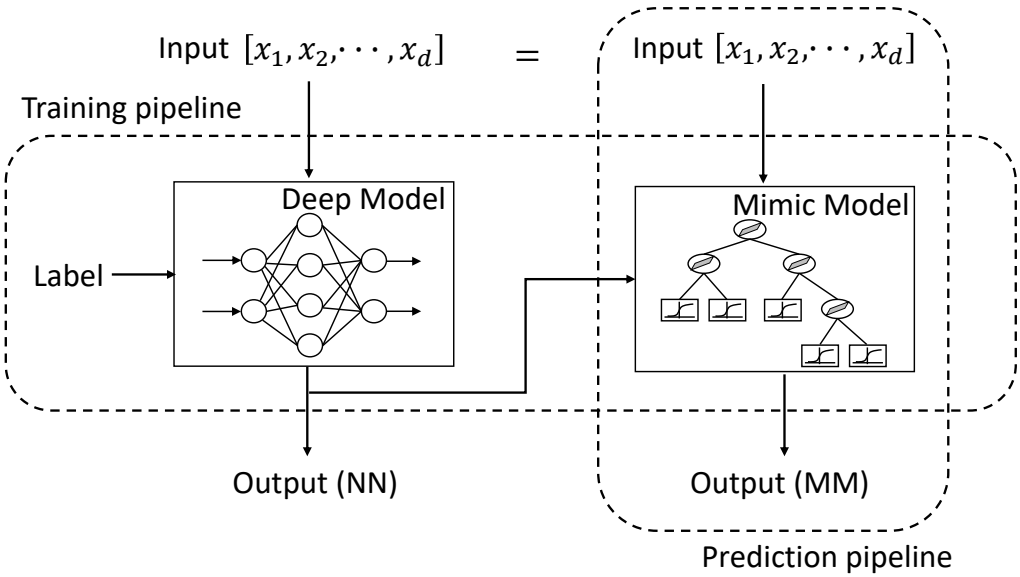


Figure 4.2: Training and prediction pipeline of mimic learning.

Definition 4. *Given a deep model $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}^c$, the goal of interpreting by mimic learning task is to learn a mimic model G such that $\forall x \in \mathbb{R}^d$: $G(x) \approx \mathcal{N}(x)$. G is obtained by minimizing the expectation of the specified objective function $\mathcal{L}(\mathcal{N}(x), G(x))$ over all x values:*

$$\arg\min_{G} \mathcal{L}(\mathcal{N}(x), G(x))$$

where $\mathcal{L}(\mathcal{N}(x), G(x))$ is devised to measure the approximation error.

In order to generate interpretations of the target model, the mimic model G is expected to be with a simple and interpretable model structure and competitive prediction performance. The tree structure is a typical machine learning model structure which is usually with good interpretability. Therefore, we propose a tree structure to mimic the given target model, named the oblique model tree.

4.3 Interpretable Oblique Model Tree

In this section, we propose our solution to interpret a given black-box deep model by mimicking an oblique model tree from the deep model. Our introduction includes three parts. First, we introduce our proposed oblique model tree structure. Then we propose the mimic learning algorithm for learning an oblique model tree from the given deep model. Finally, we discuss the interpretations given by the mimicked oblique model tree.

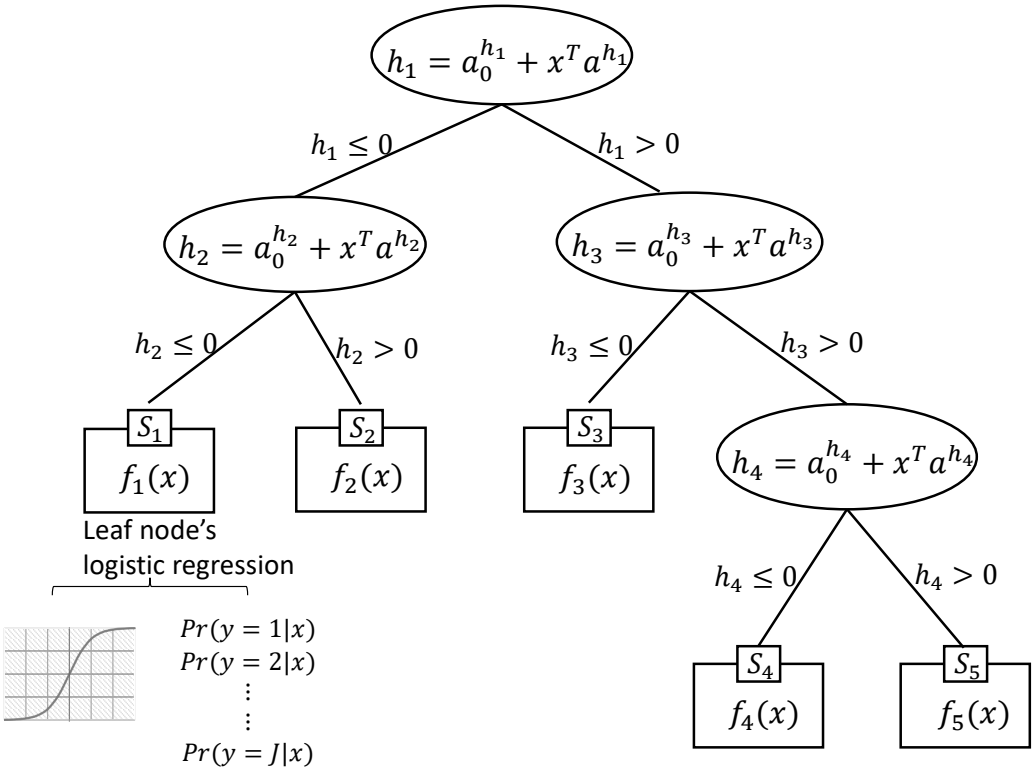


Figure 4.3: An example of the oblique model tree

4.3.1 Oblique Model Tree

The oblique model tree (OMT for short) is an extraordinary binary decision tree structure. Each internal node contains an oblique split which is a linear combination of input features. Each leaf node corresponds to a classification model that is constructed using logistic regression.

Formally, given a classification task, let X denote the set of input instances, let $x = \{x_1, x_2, \ldots, x_d\}$ denote an arbitrary input instance in X that consists of d features , d denotes the dimension of features. Instances in X are classified into c class labels. Let y denote the class label of x, we have $y \in \{1, 2, \ldots, c\}$. An oblique model tree built on X consists of a set of non-leaf nodes P and a set of leaf nodes T.

For an arbitrary non-leaf node $p \in P$, there is an oblique split on p, denoted by h_p. The oblique split h_p is a linear combination of the d-dimensional features, written as

$$h_p(x) = a_0 + \sum_{i=1}^{d} a_i x_i \tag{4.1}$$

where $a = \{a_0, a_1, a_2, \ldots, a_d\}$ denotes the coefficients of h_p. The division of child nodes is based on the calculation of $h_p(x)$ comparing with zero. Specifically, an input instance belonging to left child should satisfy $h_p(x) \leq 0$, an input instance belonging to right child node should satisfy $h_p(x) > 0$.

For a leaf node $t \in T$, a corresponding logistic regression classifier $f_t(x)$ is constructured to make predictions. The logistic regression classifier $f_t(x)$ is written as

$$f_t(x) = \arg\max_i \{Pr(y = i|x)\} \tag{4.2}$$

$$Pr(y = i|x) = \frac{e^{\ell_i(x)}}{\sum_{k=1}^{c} e^{\ell_k(x)}} \tag{4.3}$$

$$\ell(x) = \begin{pmatrix} w^{(1)} \\ w^{(2)} \\ \vdots \\ w^{(c)} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} \tag{4.4}$$

where $w^{(i)} = \{w_0^{(i)}, w_1^{(i)}, \ldots, w_d^{(i)}\}$ denotes the weight vector corresponding to i-th label, $\ell_i(x)$ is the i-th item in $\ell(x)$, $Pr(y = i|x)$ denotes the probability of x being predicted to label i. $f_t(x)$ finally prints out the label with maximum predicted probability.

Let us consider the effectiveness of an oblique model tree in the input space. Let S represent the whole input space. S is divided by the oblique model tree into a set of disjoint partitions, where each internal node $p \in P$ of the tree corresponds to a space splitting hyperplane and each leaf node $t \in T$ represents a subregion S_t. That is,

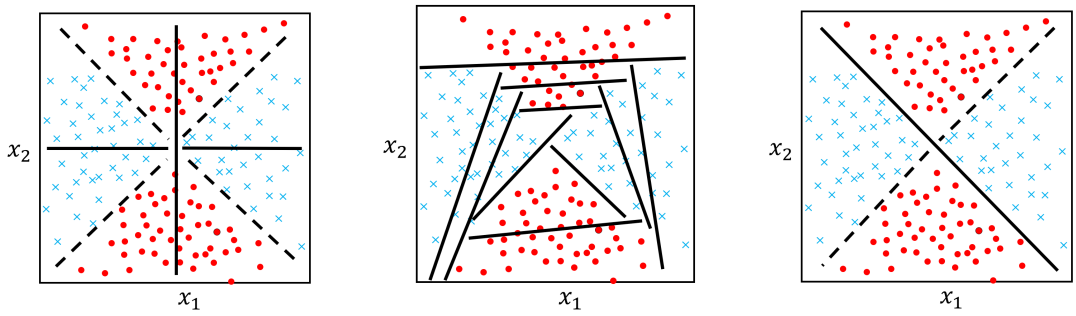
$$S = \bigcup_{t \in T} S_t, \quad S_t \cap S_{t'} = \emptyset \text{ for } t \neq t' \tag{4.5}$$

In this case, the mimic model G which is an oblique model tree can be written as

$$G = \sum_{t \in T} c_t f_t(x) \tag{4.6}$$

where $c_t = 1$ if $x \in S_t$, otherwise we have $c_t = 0$.

Figure 4.3 shows an example of the oblique model tree. Starting from the root node, the oblique splitting hyperplane h_1 at root node divides the input space into two half subspaces. Then another two oblique splitting hyperplanes h_2, h_3 act on two half subspaces separately to do further splitting. h_4 acts on a half subspace of h_3. Finally, the whole input space is



(a) LMT (b) ODT (c) OMT

Figure 4.4: An example to illustrate how the proposed algorithm works. (a) the axis-parallel splits and leaf LR model learnt by logistic model tree, (b) the oblique splits learnt by oblique model tree, using the OC1 algorithm[87, 93], (c) the oblique splits and leaf LR model learnt by our method.

divided by this oblique model tree into five small subregions, denoted by $S_1, S_2 \ldots S_5$. Each subregion corresponds to a leaf node of the tree. The logistic regression classifier at each leaf node does approximation to the behavior of the deep model in the corresponding local region.

An oblique model tree represents a piecewise linear approximation to the given deep model. It divides the input space into a number of disjoint regions by oblique hyperplanes $p \in P$. These oblique hyperplanes make the oblique model tree more flexible than logistic model tree structures [73]. That is, intuitively the number of disjoint regions divided by an oblique model tree is always smaller than regions generated by a logistic model tree, on the same task. In addition, an oblique model tree generates a logistic regression classifier f_t within each linear region $S_t \in S$, and can deal with more complicated regions than the traditional oblique decision tree. In this case, when dealing with the same prediction task, the number of regions divided by an oblique model tree is intuitively smaller than that of the oblique decision tree. Figure 4.4 gives an example to illustrate the advantages of the oblique model tree over the oblique decision tree and the logistic model tree.

To classify a given input x, we start from the root node of the tree and search for a path from the root node to a specific leaf node. This leaf node points out the subregion to which x belongs. To find this path, at each internal node p, we compare x with the current splitting hyperplane $h_p(x)$, then descend to the left child if $h_p(x) \leq 0$, otherwise to the right child. After reaching a certain leaf node, the prediction is given by the logistic regression model.

4.3.2 Building Oblique Model Tree

We devise an algorithm to build an oblique model tree recursively starting from the root node.

Each internal node of an oblique model tree represents a splitting hyperplane that divides the corresponding input space region into two halves. Two child nodes correspond to two disjoint subregions divided by the splitting hyperplane. If two child nodes are leaf nodes, an individual logistic regression classifier is built to predict instances within each subregion. In order to divide the tree nodes and stop dividing when appropriate, we temporally treat two child nodes as leaf nodes when dividing a node. In this case, two problems need to be solved during the node division, that is, determining the space splitting hyperplane and determining the logistic regression classifier on each child node.

To divide a tree node, we start by initializing a splitting hyperplane randomly. After that, the idea of Lloyd's algorithm [82] is adopted to update the splitting hyperplane and build the child nodes' logistic regression classifiers. Specifically, the following two steps are executed iteratively:

1) Divide the input space according to the current splitting hyperplane $h_p(x)$, then optimize the logistic regression model on each child node.

2) Update the splitting hyperplane based on the logistic regression models on child nodes computed in Step 1.

The objective function $\mathcal{L}(\mathcal{N}, G)$ is designed to reflect the approximation degree of G to $\mathcal{N}$. The approximation degree on a tree node p is written as

$$\mathcal{L}_p(\mathcal{N}, G) = -\frac{1}{|X_p|} \sum_{x \in X_p} \sum_{i=1}^{c} \hat{y}_i \log p(y = i|x) \tag{4.7}$$

where X_p denotes the set of input instances at node p, $p(y = i|x)$ denotes the probability of x being predicted by the oblique model tree to label i, G denotes the oblique model tree whose root node is p. We formulate the prediction probability $p(y = i|x)$ as a combination of space spliting and the prediction capability of the child nodes. Specifically, $p(y = j|x)$ represents the combined probability of x belonging to label j, and is calculated by combining the probability of x being divided to each child node multiply the prediction probability given by the logistic regression model on corresponding child node, written as

$$p(y = i|x) = p(x \in X_l) Pr(y = i|x, X_l) + p(x \in X_r) Pr(y = i|x, X_r) \tag{4.8}$$

where $Pr(y = i|x, X_l), Pr(y = i|x, X_r)$ are computed by the logistic regression classifier on left, right child node, respectively (Equation 4.3), X_l, X_r are smaller sets of instances at two child nodes of p, divided by $h_p(x)$ and satisfy $X_p = X_l \cup X_r$.

The *sigmoid* function is applied on $h_p(x)$ to calculate the probability of x belonging to each child node, written as

$$\begin{aligned} p(x \in X_r) &= sigmoid(h_p(x)), \\ p(x \in X_l) &= 1 - p(x \in X_r) \end{aligned} \tag{4.9}$$

Both the splitting hyperplane and the logistic regression models of child nodes are optimized using the gradient descent approach. An overview of OMT building algorithm is shown in Algorithm 2.

Our designed algorithm constructs the oblique model tree in order from simple to complex model. The size of the oblique model tree relies on the complexity of the original deep model and the complexity of the training dataset. For simple datasets that can even be classified by logistic regressions to achieve good prediction performance, our algorithm would prefer to stop the growth of the oblique model tree at the root node, thus it is simplified to a standard logistic regression model. For large and complex datasets, a well-designed oblique model tree with a larger tree size will be constructed.

The time complexity of building an oblique model tree with tree depth h is approximate $O(chnd^2)$, where n is the number of samples in the input dataset assigned to this node, and d is the number of attributes. As analyzed in [73], the runtime of a simple logistic regression is $O(nd^2)$. If the maximum number of the outer iteration(Algorithm 2, line 6) is taken as a constant c, the time complexity of constructing a node is $O(cnd^2)$.

Stop condition. The algorithm is recursively run from the root node of an oblique model tree until at least one of the following termination conditions is satisfied:

(1) the number of samples assigned to the node is less than a threshold. This condition is because the logistic regression models at leaf nodes need enough samples for model fitting;

(2) the prediction accuracy of the current node is higher than a threshold, which is our expected approximating capability;

(3) the node splitting is seriously unbalanced. That is, a child node has much more samples than the other one. Stop splitting such a node can avoid the occurrence of overfitting;

Algorithm 2 BuildingOMT

Require: The current input set X, a deep model N

Ensure: A oblique model tree T

{Initialization}

$a \leftarrow$ parameter vector of oblique split

$w_l \leftarrow$ weight vector of left child's logistic model

$w_r \leftarrow$ weight vector of right child's logistic model

$w \leftarrow \{w_l, w_r\}$

$\epsilon \leftarrow$ convergence threshold of $\mathcal{L}$

repeat

$X_l \leftarrow \{x_i | h(x_i) \leq 0, x_i \in X\}$

$X_r \leftarrow X - X_l$

$\mathcal{L}_{prev} \leftarrow \mathcal{L}$

{Optimize logistic model on each child node using gradient descent}

repeat

current $\mathcal{L} \leftarrow cost(X, N(X), a, w)$

$w \leftarrow w - \lambda_w \nabla \mathcal{L}$

until $\mathcal{L}$ convergence

{Optimize oblique split using gradient descent}

repeat

current $\mathcal{L} \leftarrow cost(X, N(X), a, w)$

$a \leftarrow a - \lambda_a \nabla \mathcal{L}$

until $\mathcal{L}$ convergence

until $|\mathcal{L} - \mathcal{L}_{prev}| < \epsilon$

if termination conditions are not satisfied **then**

$T_l \leftarrow BuildOMT(X_l, N)$

$T_r \leftarrow BuildOMT(X_r, N)$

Attach T_l, T_r to the corresponding branch of T

end if

return T

(4) the node splitting causes a reduction of energy compared to the classification model of the node itself. This demonstrates that the node splitting has no positive increase in prediction performance.

Our experimental results show that these stop criteria work well in building an oblique model tree.

Data augementation. We adopt data augmentation in building oblique model trees. During the tree constructing process, the input space is iteratively divided into smaller regions. In some small divided regions, the number of corresponding instances becomes very small. This may lead to overfitting in the corresponding regions and destroy the generalization capability. We use a standard data augmentation technique, namely Gaussian noise [2], to enlarge the data set in such regions. New instances are generated by adding random Gaussian noises to the original instances. Then the generated Gaussian noise instances are given to the oblique model tree, to filter out these located in the current region. Data Augmentation is applied to regions where the number of instances in the region lower than a fixed threshold.

4.3.3 Interpreting by Oblique Model Tree

The oblique model tree offers interpretations to predictions made by the target deep model. First, a path from the root node to a certain leaf node tells which leaf node, or which local region the input sample belongs to. Then, the logistic regression classifier on the leaf node can map the prediction result back to the feature attributions and measure the importance of feature attributions on the prediction.

The good interpretability of oblique model trees benefits from the tree structure and the local logistic regression model on each leaf node. Tree structures have good interpretability because of their clear edge-node natural structure and rule-extraction capability, in this case, the tracking of a given sample flow to a specific small region of a specific leaf is clear and distinct. The logistic regression model at the leaf node offers an intuitive interpretation to the given instance by explaining the feature importance in a prediction through the value of the weight vector.

4.4 Experimental Results

In this section, we present experiments to evaluate our algorithm. We organize this section by answering to following three questions:

Dataset	# Training	# Testing	# Features
Spambase	3,500	1,100	50
Bi-MNIST	8,000	5,000	784
Bi-FMNIST	8,000	6,000	784

Table 4.1: Statistics of data sets

- How is the performance of the oblique model tree as a classifier, comparing with deep models and other classification methods? (Section 4.4.2)
- How is the mimicability of OMT? (Section 4.4.3)
- How does OMT provide interpretation to deep models? (Section 4.4.4, 4.4.5)

These questions will be answered in the following sections, respectively.

4.4.1 Experiment Setup

In order to better present and visualize the result, our experiments are all on binary classification problems. It is easy to generalize our approach to multidimensional classification problems. We use three datasets, including one text dataset and two image datasets:

Spambase: This is a spam email text set from UCI [28]. Each instance represents an email with the label 'spam' or 'non-spam'. Each instance is represented by 57 features, such as word frequencies, statistics of character sequences. We apply simple data cleaning to remove meaningless stop words and remain 50 useful features.

Bi-MNIST: This is a subset of the hand-writing MNIST dataset [76]. The MNIST contains 10 class labels, each of which is an image group of the hand-writing number from '0' to '9'. Bi-MNIST consists of the hand-writing '5' and '6'. Each instance is a 28 by 28 grayscale image.

Bi-FMNIST: This is a subset of the Fashion-MNIST dataset [122]. The Fashion-MNIST is a set of Zalando's article images and contains 10 class labels, with each instance a 28 by 28 grayscale image. We choose data within class 'Pullover' and 'Coat' to form the Bi-FMNIST dataset.

Table 4.1 shows the statistics of datasets we used. We train deep models on these datasets by optimizing the classification performance. We train a two-hidden-layer DNN for Spam-

Methods	Spambase	Bi-MNIST	Bi-FMNIST
DT	0.916 ± 0.016	0.966 ± 0.005	0.814 ± 0.010
LR	0.925 ± 0.016	0.980 ± 0.004	0.860 ± 0.009
SVM	0.915 ± 0.016	0.987 ± 0.003	0.864 ± 0.009
AdaBoost	0.919 ± 0.016	0.982 ± 0.004	0.838 ± 0.008
OMT	0.927 ± 0.015	0.988 ± 0.003	0.879 ± 0.008
NN	0.933 ± 0.015	0.994 ± 0.002	0.905 ± 0.007

Table 4.2: The prediction accuracies of OMT, neural network, and baseline models on the test datasets.

base dataset, with the structure of $[784, 100, 18, 2]$. We train CNN models with the structure of LeNet-5 [75] for both Bi-MNIST and Bi-FMNIST.

All experiments were run on a PC with Windows 7 system, 16 GB memory, and a 3.40 GHz Intel Core i7-3770 CPU processor. All codes and methods are implemented in Matlab. The DNN model is implemented using the Matlab DeepLearnToolbox [2], two CNN models are implemented using the MatConvNet toolbox [119].

4.4.2 Prediction Performance

To demonstrate that the oblique model tree achieves comparative classification performance with the teacher deep model, we design an experiment to examine the prediction accuracy of oblique model trees. In this experiment, we consider an oblique model tree as an individual classification model. We compare the oblique model tree with several baseline methods, which are typical machine learning classification models, including Logistic Regression (LR), Decision Tree (DT), Support Vector Machine (SVM), and AdaBoost [37]. The SVM uses the radial basis function as the kernel. Baseline models are trained using the ground-truth labels. All models and parameters are trained to their best.

Table 4.2 shows the prediction performance of all these methods. The accuracy is the average of 10-fold cross-validation random trials. The results demonstrate that, deep models (DNN for Spambase, CNNs for both Bi-MNIST and Bi-FMNIST) always achieve the best accuracy on all datasets. Meanwhile, OMTs maintain strong prediction accuracy, which is very close to the deep neural networks, better than the other four baseline methods.

[2] `https://www.mathworks.com/matlabcentral/fileexchange/38310-deep-learning-toolbox`

Dataset	$d(y_{nn}, f)$	$\rho(y_{nn}, f)$
Spambase	0.019	0.911
Bi-MNIST	0.007	0.990
Bi-FMNIST	0.059	0.931

Table 4.3: Compare distance and correlation between the prediction probability given by OMT and that given by deep model on testing datasets.

This indicates that an OMT can be used as an individual classifier to achieve competitive prediction performance.

4.4.3 Mimicability

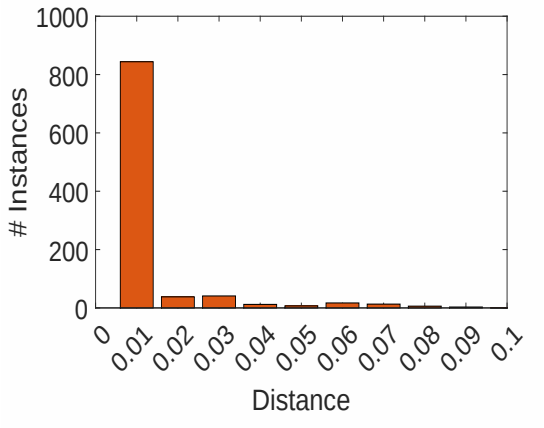
To investigate the mimicability of an oblique model tree to the deep model, we design experiments to examine how close the prediction probabilities given by an OMT to the prediction probabilities given by the target deep model. These experiments verify whether the OMT behaves similarly to the deep model. The prediction results of both the deep model and OMT are probabilities of belonging to a specific class, which is within $[0, 1]$. We propose Manhattan distance and correlation coefficient as representations of the similarity. The Manhattan distance d is in the form of

$$d(y_{nn}, f) = \sum |y_{nn} - f(x)|. \tag{4.10}$$

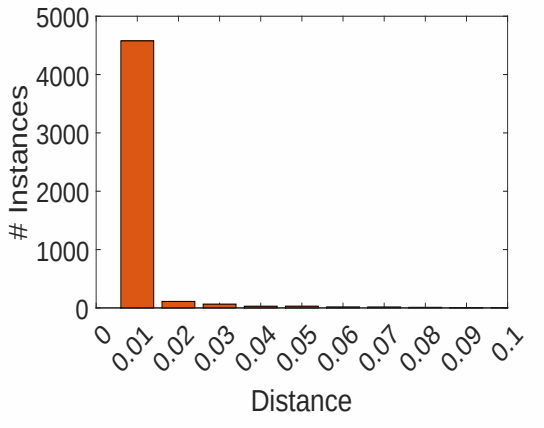
The correlation coefficient ρ is with the form of

$$\rho(y_{nn}, f) = \frac{Cov(y_{nn}, f(x))}{\sigma_{y_{nn}} \sigma_f}. \tag{4.11}$$

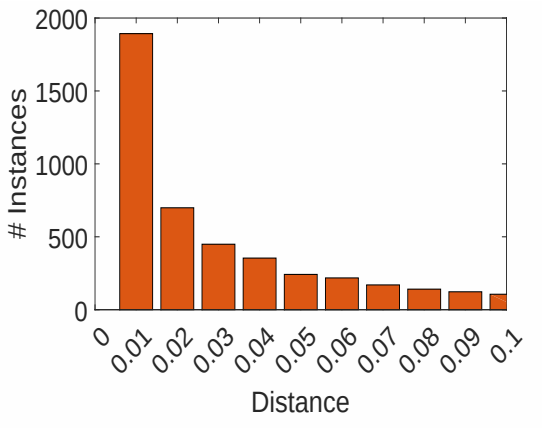
Table 4.3 shows the average Manhattan distance and correlation coefficient. These results were calculated on testing datasets of Spambase, Bi-MNIST, and Bi-FMNIST. The results show that, the average Manhattan distances on Spambase and Bi-MNIST are very small, with the value $1\% \sim 2\%$. The average distance on Bi-FMNIST is about $5\% \sim 6\%$, a little higher but still very small. In addition, on all these three datasets, the very high correlation coefficients ($\geq 91\%$) demonstrate the really tight positive correlation between these two methods.



(a) Spambase



(b) Bi-MNIST



(c) Bi-FMNIST

Figure 4.5: This figure shows the distance distribution between OMTs and the teacher deep models, on three testing datasets. Here x-axis represents the distance, y-axis is the number of instances corresponding to the current distance.

The histograms in Figure 4.5 show the distributions of Manhattan distance among instances. From this figure, we observe that for the Bi-MNIST dataset, more than 90% instances are within a shorter than 0.01 distance. For the Spambase dataset, this percentage is 80%. For the Bi-FMNIST dataset, more than 70% instances are within a shorter than 0.05 distance. The extremely unbalanced distribution shows, in most cases, the prediction results obtained by the OMT are very similar to the deep model. These results consist of the main idea of OMT, that is, learn a piecewise linear approximation of the black box deep model. Instances with large distances are likely located at regions with highly-curved neural network decision boundaries.

4.4.4 Examining learned features

To demonstrate that the learned oblique model tree is able to provide good interpretations to the neural network model, we analyze the interpretations given by the oblique model tree. We design two experiments to prove that the oblique model tree is able to extract decision features for the deep model's prediction.

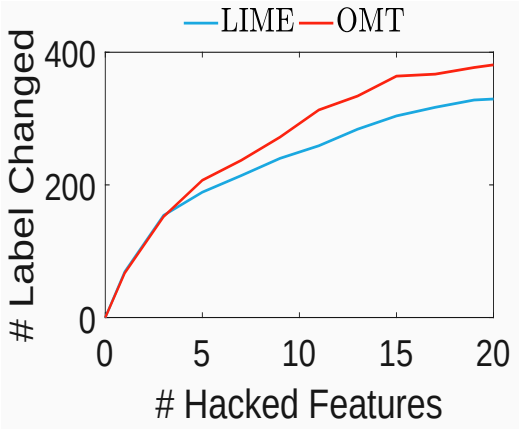
The first experiment is to **add perturbations to decision features selected by the OMT**, then examine the influences of such perturbations on prediction results. More precisely, given a certain instance x, at first, by using the OMT the top k most important decision features of x are selected. Then, small perturbations are added to these k features to generate x'. Finally, x' is set into the original neural network model to make a prediction. In our experiment, the perturbations are added by setting the values of these features to be zero, which is to directly "blind" these features. Intuitively, the larger the influence on the prediction results of the deep neural network, the more important the selected features.

In this experiment, we use LIME, a local interpretation method proposed by Ribeiro *et al.* [107] as the baseline. Experiments are designed on test datasets of Spambase, Bi-MNIST, and Bi-FMNIST, and results are shown in Figure 4.6. In Figure 4.6, we count the number of instances going opposite prediction label after adding perturbations. The blue line corresponds to hack top important features selected by LIME while the red line corresponds to hack important features selected by OMT. Besides, Figure 4.7 shows the influence on the deep model's prediction accuracy after adding perturbations. From these figures, we can see that in all three datasets, the selected features of OMT have a greater impact than LIME, which indicates that OMT has learned better features from the deep model.

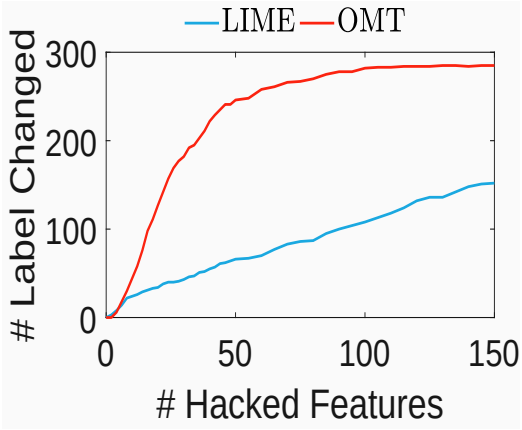
The second experiment is to **investigate whether OMT can learn decision boundaries of neural networks well**. Given the input instance x, $r(x)$ is the vector with minimal $||r||_2$ such that $x + r(x)$ locates at the decision boundaries of the deep model. That is, $r(x)$ is expected to be orthogonal to the decision boundaries of the deep model at $x + r(x)$ and $||r(x)||_2$ represent the minimal distance of x to the decision boundaries of the deep model. We follow the idea of Fawzi *et al.* [35] and Szegedy *et al.* [118] to approximate the value of $r(x)$ by finding the smallest perturbations required to misclassify x. This idea is known as the adversarial perturbation [101]. In this case, $r(x)$ is approximated by optimizing

$$\arg\min \|r\|_2 \quad s.t. \quad \hat{y}(x + r) \neq \hat{y}(x). \tag{4.12}$$

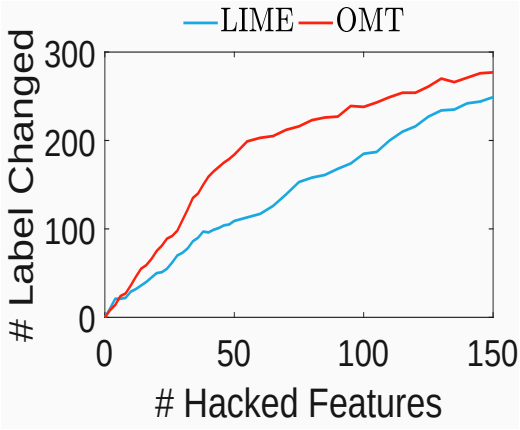
The algorithms from [35, 118] are used to approximate $r(x)$.



(a) Spambase



(b) Bi-MNIST

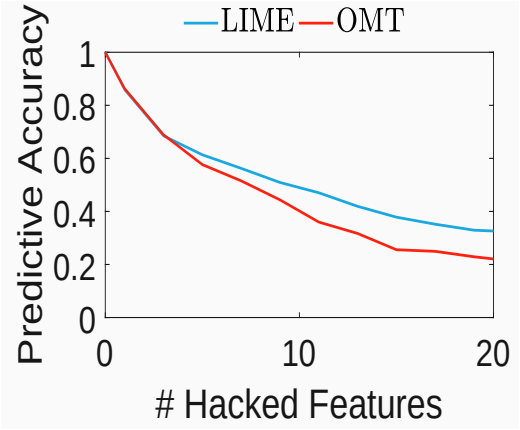


(c) Bi-FMNIST

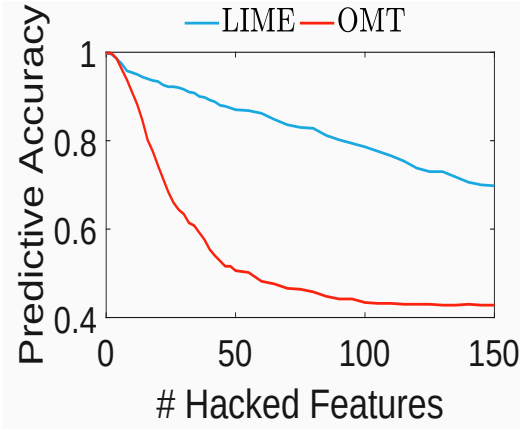
Figure 4.6: This figure shows the number of instances been flipped by adding perturbation. Here x-axis represents k, the number of features been perturbed, y-axis shows the number of instances being predicted to opposite labels.

The logistic regression model in each leaf node of an OMT provide an approximation of the deep model's decision boundaries within the corresponding small region. Specifically, $(w^{(i)} - w^{(j)})x + (w_0^{(i)} - w_0^{(j)}) = 0$ (Eq. 4.4) represents the decision boundary between class label i and j within the small region. Here $(w^{(i)} - w^{(j)})$ provide the direction perpendicular to this mimicked decision boundary. In particular, in our experiments there are only 2 class labels, thus we use w to represent $(w^{(1)} - w^{(0)})$.We define $r_{omt}(x)$ as the vector with the minimal distance of x to the decision boundary of the OMT, and have

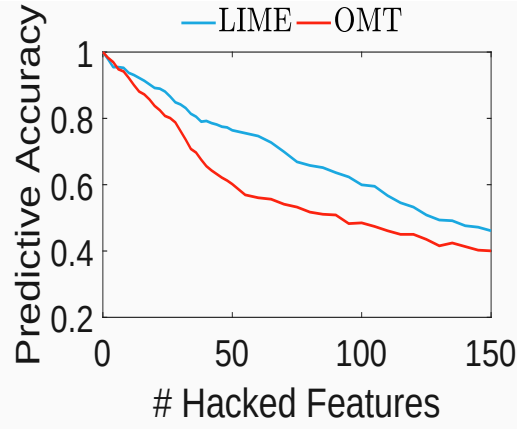
$$r_{omt}(x) = a * \frac{w}{\|w\|_2} \tag{4.13}$$



(a) Spambase



(b) Bi-MNIST

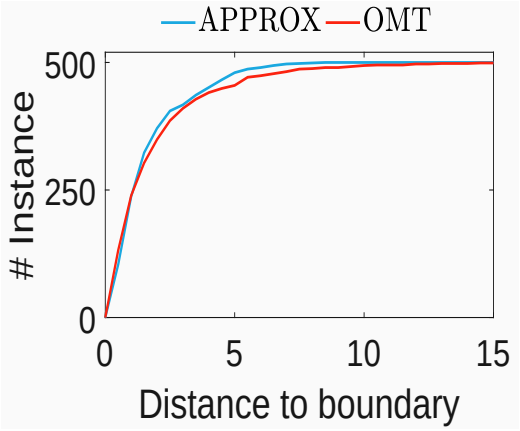


(c) Bi-FMNIST

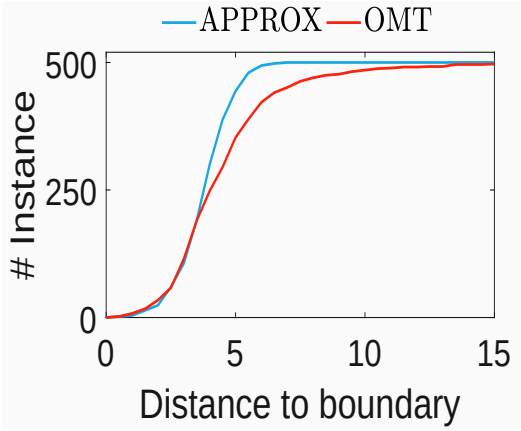
Figure 4.7: This figure shows the changes in the deep model's prediction accuracy after perturbation. Here x-axis is k, the number of features been perturbed, and y-axis represents the accuracy after perturbing.

where a is computable.

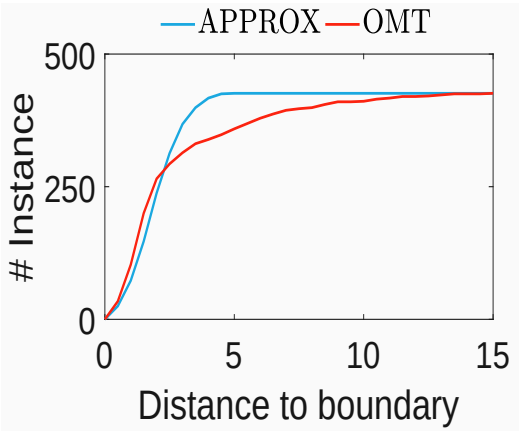
We compare $||r(x)||_2$, the minimal distance of x to the decision boundaries of the deep model, with $||r_{omt}(x)||_2$, the minimal distance of x to the decision boundaries of the learned OMT. Figure 4.8 shows the distribution of the minimal distances to the decision boundaries. This experiment is implemented on 500 input instances which are randomly selected from testing datasets. The results show that, on all three datasets, the statistical distributions of $\|r_{omt}(x)\|_2$ is very close to the distributions of $\|r(x)\|_2$, which indicates that the decision boundaries of the learned OMT lie very close to the decision boundaries of the original neural network.



(a) Spambase



(b) Bi-MNIST



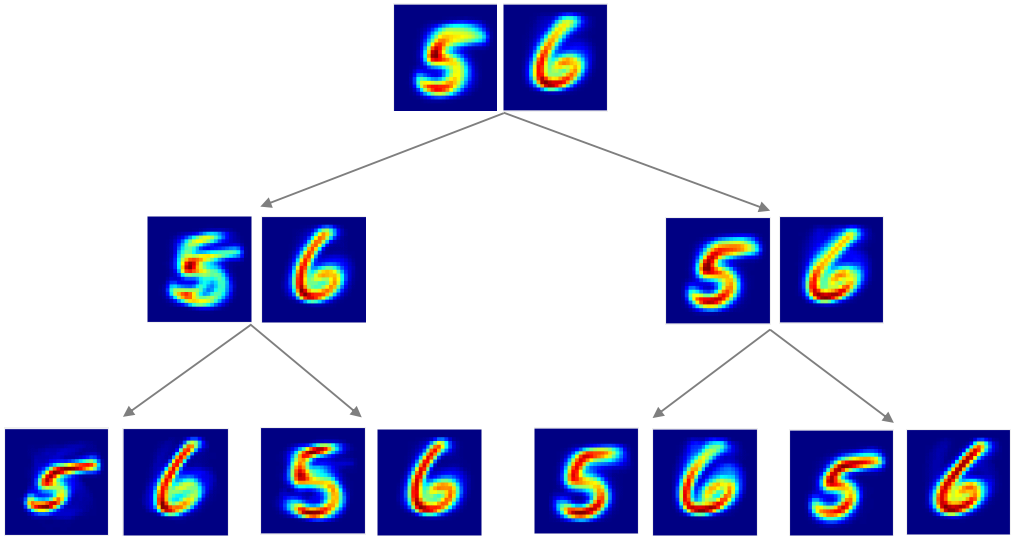
(c) Bi-FMNIST

Figure 4.8: This figure shows distributions of the minimal distances to the decision boundaries. The legend label APPROX represents $||r(x)||_2$ and OMT represents $\|r_{omt}(x)\|_2$.

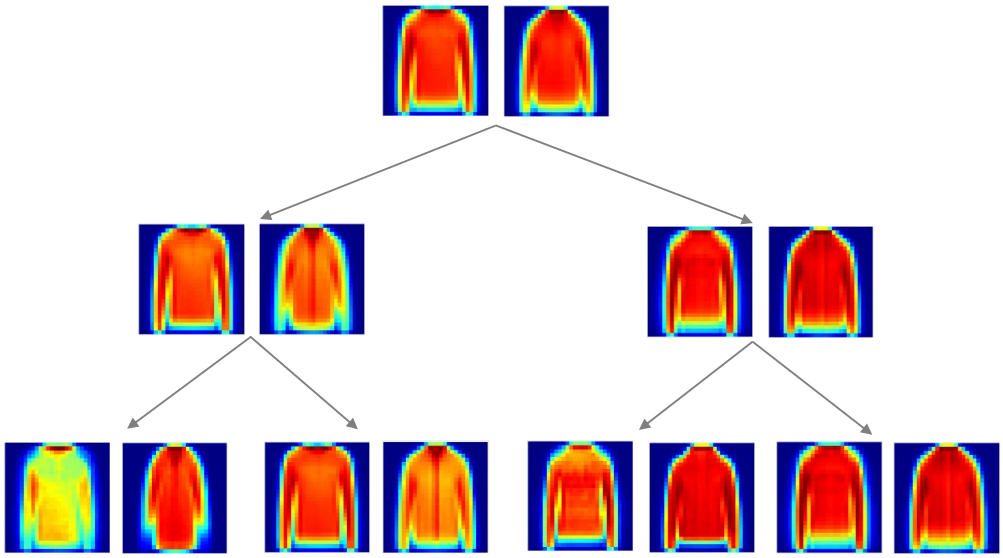
4.4.5 Feature explanation

Interpretations of deep neural networks are expected to help humans understand how the model works. The oblique model tree interprets the original neural network model by telling which are the decision features for a specific prediction result. In this experiment, we investigate how these decision features selected by OMT meet with human understanding. We provide both the global view and instance view explanations.

Global view proposes to understand the entire model and provides reasons for most predictions. A well approximated OMT can be used to understand the original deep model, specifically, the decision boundaries in the input space, the main rule extractions for making predictions. In Figure 4.9 and 4.10, we visualize the global view of the learned OMT structure, where images upon each node represent the average image corresponding to each



(a) Bi-MNIST



(b) Bi-FMNIST

Figure 4.9: Examples of OMT splitting input space on Bi-MNIST and Bi-FMNIST datasets. In (a)(b), two figures on each node represent the average images of instances with each label.

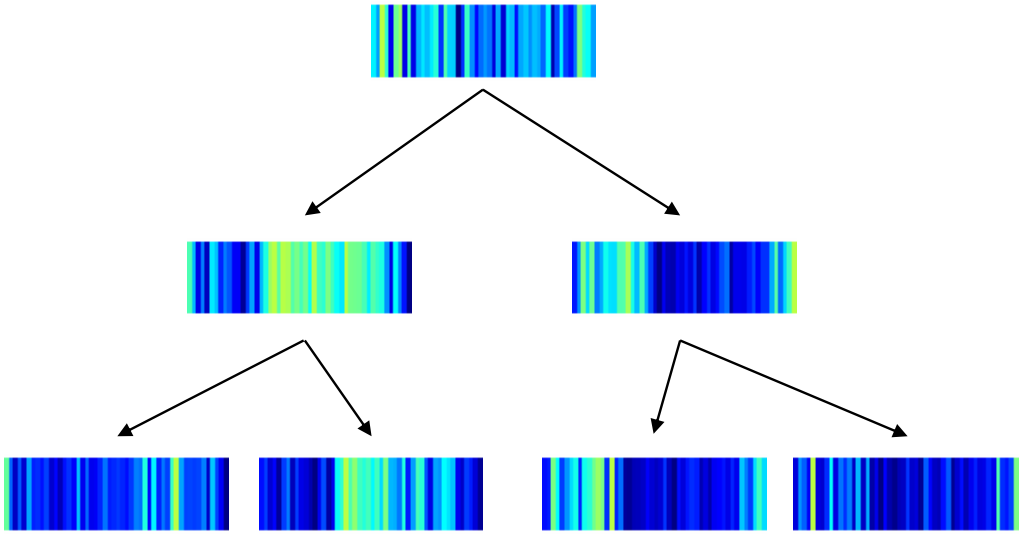
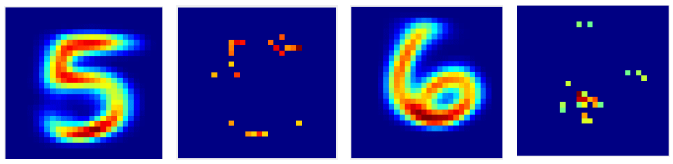


Figure 4.10: Examples of OMT splitting input space on Spambase dataset. We group features of Spambase into visible spectrum where the brighter the position is, the greater the word frequency.

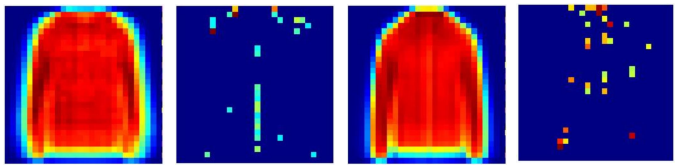
label, flowing through this node. Compare the images of a node and images of the child nodes shows how the dataset is divided into two parts and flow to both child nodes. For instance, in Figure 4.9(a) the average shape of handwriting '5' and '6' of the current node are shown on each node. The results show that, the node splittings mostly depend on the shapes of images. When the tree grows from root to deeper layers, the average images are with clearer characteristics and shapes.

Figure 4.11 shows examples of regional classification provided by the logistic regression models on leaf nodes. In figure 4.11 (a) and (b) the four figures from left to right are the average images of the first label (i.e. handwriting '5', class 'pullover'), its decision features provided by OMT, the average image of the other class label (i.e. handwriting '6', class 'coat'), and the decision features provided by OMT for recognizing this class. Figure 4.11(c) shows decision features given by OMT for predicting spam and non-spam email in a certain leaf region. We can see from these figures that OMTs extracted decision features locates at most recognizable positions, which is with our expectation, such as in figure 4.11(b) for recognizing 'pullover', the wide shoulder and zipper[3] are extracted, for recognizing 'coat' the tall collar is extracted.

[3]Pullover has no zipper, so the zipper position is with deeper color than the coat.



(a) Bi-MNIST

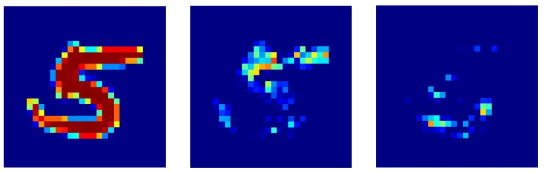


(b) Bi-FMNIST

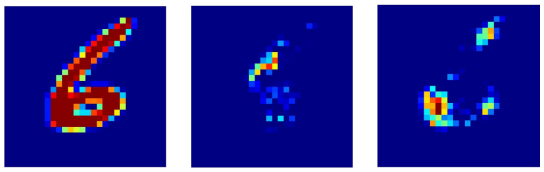
Most Freq Words	email	addresses	remove	capital_length	receive	mail
Spam	$	credit	business	remove	addresses	money
Non-spam	edu	cs	original	internet	order	meeting

(c) Spambase

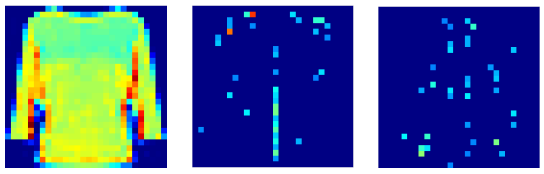
Figure 4.11: In this figure, (a)(b) show the average images and OMT's decision features of typical local regions of Bi-MNIST and Bi-FMNIST, (c) shows the most frequency words and OMT's selected decision words in a local region of Spambase.



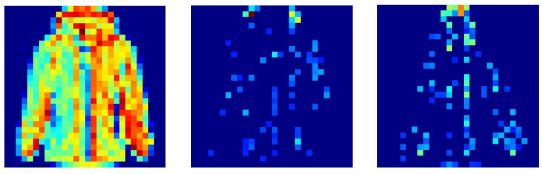
(a) Bi-MNIST '5'



(b) Bi-MNIST '6'



(c) Bi-FMNIST 'Pullover'



(d) Bi-FMNIST 'Coat'

Figure 4.12: Individual instance analysis. In each subfigure, the first image is the original instance image, the second and third images represent the decision features of image being classified to each label.

Instance view provide interpretations to specific predictions. Given an arbitrary input instance x, the OMT is able to provide interpretations to the prediction result of x. Figure 4.12 visualizes the interpretations, which are feature importance. The instances are arbitrarily selected from each class label. The three images in each subfigure from left to right are the chosen instance, the $w \cdot x$ distribution for first label class (i.e. '5' and 'pullover'), the $w \cdot x$ distribution for the other label class (i.e. '6' and 'coat'), respectively. This figure intuitively displays why a specific instance is classified to a certain label. For example, in figure 4.12(a) the $w \cdot x$ distribution of class '5' is brighter and larger than the distribution of class '6', which means it 'align' more with the recognizable position of handwriting '5' compared with handwriting '6'.

4.5 Summary

In this chapter, we study the interpretation of deep neural networks through mimic learning. First, we propose the oblique model tree, which is a simple, interpretable tree model and can mimic a given deep model. Then we introduce an algorithm to build an oblique model tree from the given deep neural network. We discuss the interpretations given by a mimic oblique model tree to the deep neural network. Our experimental results on image and text datasets show that our proposed oblique model tree achieves comparative prediction performance with the deep model, and closely mimic the behaviors of the deep model. We also show that an oblique model tree can provide reasonable interpretations to predictions made by the target deep model.

There are several follow-up directions of this project worthy of further study. The first is to develop a more effective and faster algorithm to build oblique model trees. We show in this Chapter that an oblique model tree is able to approximation a black-box deep model to give an interpretation and is with high expressive capacity and good interpretability. Thus, an efficient algorithm can decrease the time cost to build an oblique model tree thus help the application of the oblique model tree in many problems. Besides, the applications of oblique model trees are not restricted to the deep model interpretation problem, it is also interesting to apply the oblique model tree structure to other applications. In particular, the oblique model tree can be used as an individual prediction model. If the logistic regression model is used on the leaf nodes, it becomes a classification model. Different local models can be used on leaf nodes when applying to specific applications.

Chapter 5

Piecewise Linear Approximation of Deep Curve Neural Networks for Model Complexity

Model complexity is a fundamental measure for deep neural networks. A good model complexity measure can help to tackle many challenging problems, such as overfitting detection, model selection, and performance improvement. The existing literature on model complexity mainly focuses on neural networks with piecewise linear activation functions. Model complexity of neural networks with general smooth curve activation functions remains an open problem. In this chapter, we develop a complexity measure for deep fully-connected neural networks with smooth curve activation functions. To achieve the measure, we first propose *linear approximation neural network* (*LANN* for short), a piecewise linear framework to approximate a given deep model with smooth curve activation function. LANN is designed to construct individual piecewise linear approximation for activation function of each neuron, and minimize the number of linear regions to satisfy a required approximation degree. Then, we analyze the upper bound of the number of linear regions formed by LANNs, and define the complexity measure facilitated by the upper bound. To examine the usefulness of the complexity measure, we experimentally explore the training process of neural networks and detect overfitting. Our results demonstrate that occurrence of overfitting is positively correlated with the increase of model complexity during training. We find that the L^1 and L^2 regularizations suppress the increase of model complexity. Further, we propose two approaches to prevent overfitting by directly constraining model complexity: neuron pruning and customized L^1 regularization. Finally, we show that, by studying a piecewise linear approximation, LANN can directly provide interpretations for the curve neural networks using *OpenBox*.

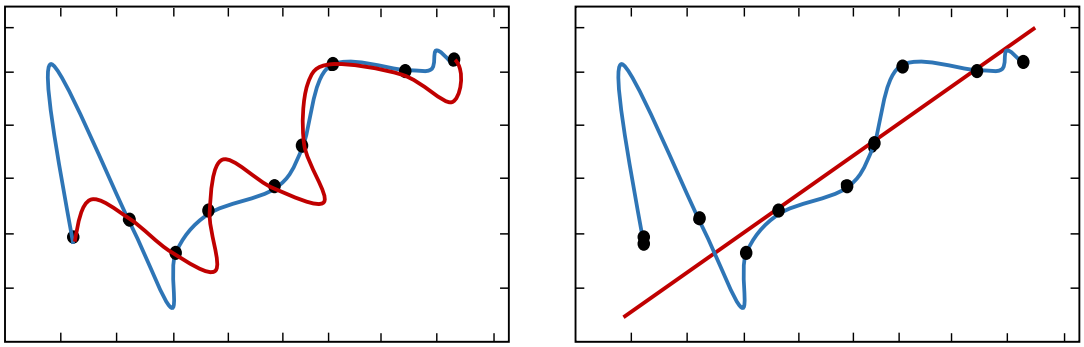
5.1 Introduction

Deep neural networks have gained great popularity in tackling various real-world applications, such as machine translation [120], speech recognition [20] and computer vision [41]. One major reason behind the great success is that the classification function of a deep neural network can be highly nonlinear and express a highly complicated function [8]. Consequently, a fundamental question asks, how nonlinear and how complex the classification function of a deep neural network is. Model complexity measures [91, 104] address this question. Progress in model complexity measure directly facilitates the advances of many directions of deep neural network, such as model architecture design, model selection, performance improvement [54], and overfitting detection [53].

The challenges in measuring model complexity are tackled from different angles. For example, the influences of model structure on complexity have been investigated, including layer width, network depth, and layer type. The power of width is discussed and a single hidden layer network with a finite number of neurons is proved to be an universal approximator [6, 61]. With the exploration of deep network architectures, some recent studies pay attention to the effectiveness of deep architectures in increasing model complexity, known as depth efficiency [8, 23, 33, 83]. The bounds of model complexity of some specific model structures are proposed, from sum-product networks [27] to piecewise linear neural networks [91, 100].

Model parameters (e.g., weight, bias of layers) also play important roles in model complexity. For example, $f_1(x) = ax + b\sin(x)$ may be considered more complex than $f_2(x) = cx + d$ according to their function forms. However if the parameters of the two functions are $a = 1$, $b = 0$, $c = 1$, and $d = 0$, f_1 and f_2 are then two coincident lines. This example demonstrates the importance of model parameters on complexity. Raghu *et al.* [104] propose a complexity measure for neural networks with piecewise linear activation functions by measuring the number of linear regions through a trajectory path between two instances. Their proposed complexity measure reflects the effect of model parameters to some degree.

However, the approach of [104] cannot be directly generalized to neural networks with smooth curve activation functions, such as Sigmoid [43, 67], Tanh [66]. At the same time, in some specific applications, smooth curve activation functions are found superior than piecewise linear activation functions. For example, many financial models use Tanh rather than ReLU [29]. A series of state-of-art studies speed up and simplify the training of neural networks with smooth curve activation functions [64]. This motivates our study on model complexity of deep neural networks with smooth curve activation functions.



(a) (b)

Figure 5.1: (a) Two functions behaving the same on a set of data points may still be very different. (b) Illustration of overfitting.

In this chapter, we develop a complexity measure for deep fully-connected neural networks with smooth curve activation functions. Previous studies on deep models with piecewise linear activation functions use the number of linear regions to model the nonlinearity and measure model complexity [91, 97, 100, 104]. To generalize this idea, we develop a piecewise linear approximation to approach target deep models with smooth curve activation functions. Then, we measure the number of linear regions of the approximation as a indicator of the target model complexity. The piecewise linear approximation is designed under two desiderata. First, to guarantee the approximation degree , we require an approximation of the function of the target model rather than simply mimicking the behavior or performance, such as the mimic learning approach [5, 60]. The rationale is that two functions having the same behavior on a set of data points may still be very different, as illustrated in Figure 5.1(a). Therefore, approximation using the mimic learning approach [5, 60] is not enough. Second, to compare the complexity values of different models, the complexity measure has to be principled. The principle we follow is to minimize the number of linear regions given an approximation degree threshold. Under these two desiderata, the minimum number of linear regions constrained by a certain approximation degree can be used to reflect the model complexity.

Technically we propose the *linear approximation neural network* (*LANN* for short), a piecewise linear framework to approximate a target deep model with smooth curve activation functions. A LANN shares the same layer width, depth and parameters with the target model, except that it replaces every activation function with a piecewise linear approximation. An individual piecewise linear function is designed for the activation function on every neuron to satisfy the above two desiderata. We analyze approximation degree of LANNs

with respect to the target model, then devise an algorithm to build LANNs to minimize the number of linear regions. We provide an upper bound to the number of linear regions formed by LANNs, and define the complexity measure facilitated by the upper bound.

To demonstrate the usefulness of the complexity measure, we analyze the training process of deep models, especially the problem of overfitting [53]. Overfitting occurs when a model is more complicated than the ultimately optimal one, and thus the learned function fits too closely to the training data and fails to generalize, as illustrated in Figure 5.1(b). Our results show that the occurrence of overfitting is positively correlated to the increase of model complexity. Besides, we observe that regularization methods for preventing overfitting, such as L^1, L^2 regularization [45], constrain the increase of model complexity. Based on this finding, we propose two simple and effective approaches for preventing overfitting by directly constraining the growth of model complexity.

The rest of the chapter is organized as follows. In Section 5.2 we provide the problem formulation. In Section 5.3 we introduce the linear approximation neural network framework. In Section 5.4 we develop the complexity measure. In Section 5.5 we explore the training process and overfitting in the view of complexity measure. In Section 5.6 we investigate the interpretation given by linear approximation neural networks. Section 5.7 concludes the chapter.

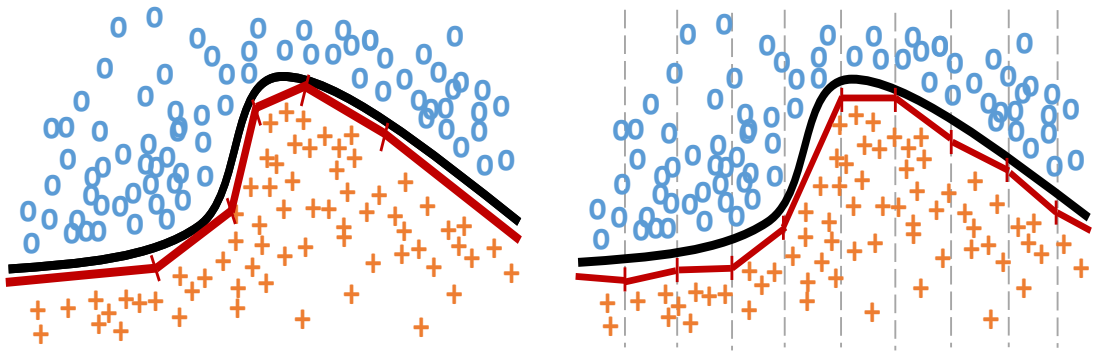
5.2 Problem Definition

A **deep (fully connected) neural network** (**DNN** for short) consists of a series of fully connected layers. Each layer includes an affine transformation and a nonlinear activation function. In classification tasks, let $f : \mathbb{R}^d \to \mathbb{R}^c$ represent a DNN model, where d is the number of features of an input, and c the number of class labels. For an input instance $x \in \mathbb{R}^d$, f can be written in the form of

$$f(x) = V_o h_L(h_{L-1}(\cdots(h_1(x)))) + b_o \tag{5.1}$$

where V_o and b_o, respectively, are the weight matrix and the bias vector of the output layer, $f(x) \in \mathbb{R}^c$ is the output vector corresponding to the c class labels, L is the number of hidden layers, and h_i is i-th hidden layer in the form of

$$h_i(z) = \phi(V_i z + b_i), \quad i = 1, \ldots, L \tag{5.2}$$



(a) Approximation g_1 (b) Approximation g_2

Figure 5.2: Example shows piecewise linear approximation under different approximation principles.

where V_i and b_i are the weight matrix and the bias vector of the i-th hidden layer, respectively. $\phi(\cdot)$ is the activation function. In this chapter, if z is a vector, we use $\phi(z)$ to represent the vector obtained by separately applying ϕ to each element of z.

In this chapter, we are interested in fully connected neural networks with curve activation functions. We focus on two typical curve activation functions, Sigmoid [43, 67], Tanh [66]. Our methodology can be easily extended to other curve activation functions.

Given a **target model**, which is a trained fully connected neural network with curve activation functions, we want to measure the model complexity. Here, the complexity reflects how nonlinear, or how curved the classification function of the network achieves. Our complexity measure should take both the model structure and the parameters into consideration. To measure the model complexity, a main intuition is to obtain a piecewise linear approximation of the target model, then use the number of linear segments of the approximation to reflect the target model complexity. This idea is inspired by the previous studies on deep models with piecewise linear activation functions [91, 97, 100, 104]. Those studies demonstrate that the number of linear segments reflects the degree of nonlinearity, that is, the complexity of deep models. To make the idea of measuring by approximation feasible, the approximation should satisfy two requirements.

First, *the quality/degree of approximation should be guaranteed.* To make the idea of measuring complexity by the nonlinearity of approximation feasible, a prerequisite is that the approximation should be highly close to the function of the target model. In this case, the mimic learning approach [5, 60], which approximates by learning a student model under the guidance of the target model outputs, is not suitable, since it learns the behavior of the target model on a specific dataset and cannot guarantee the generalizability, as il-

lustrated in Figure 5.1(a). To ensure the closeness of the approximation functions to the target models, we propose *linear approximation neural network* (LANN). A LANN is an approximation model that builds piecewise linear approximation to every activation function in the target model. To make the approximation degree controllable and flexible, we design an individual approximation function for the activation function on every neuron separately according to their status distributions (Section 5.3.1). Furthermore, we define a measure of approximation degree in terms of approximation error and analyze through error propagation (Section 5.3.2).

Second, *the approximation should be constructed in a principled and universal manner.* To understand the rationale of this requirement, consider an example in Figure 5.2, where the target model is a curved line (the solid curve). One approximation g_1 (the red line in Figure 5.2(a)) is built using as few linear segments as possible. Another approximation g_2 (the red line in Figure 5.2(b)) evenly divides the input domain into small pieces and then approximates each piece, say, using linear regression. Both of them can approximate the target model to a required approximation degree and can reflect the complexity of the target model. However, we should not use g_1 on some occasions and use g_2 on some other occasions to measure the complexity of the target model, since they are built following different protocols. To make the complexity measure comparable, the approximation should be constructed under a consistent protocal. We suggest constructing approximations under the protocal of using as few linear segments as possible (Section 5.3.3). In this case, the minimum number of linear segments required to satisfy the approximation degree can reflect model complexity.

5.3 LANN Architecture

In order to achieve a complexity measure, we propose a piecewise linear approximation to the target model. We first introduce the architecture of the linear approximation neural network. Then, we discuss the degree of approximation. Last, we propose the algorithm of building a piecewise linear approximation network.

5.3.1 Linear Approximation Neural Network

The classification function of a deep model with piecewise linear activation functions is still piecewise linear and has a finite number of linear regions [91]. The number of linear regions of such a deep model is commonly used to assess the nonlinearity of the model, i.e., the complexity [91, 97, 100, 104]. Motivated by this, we develop a piecewise linear approximation of the classification function of a target model that uses curve activation functions, then,

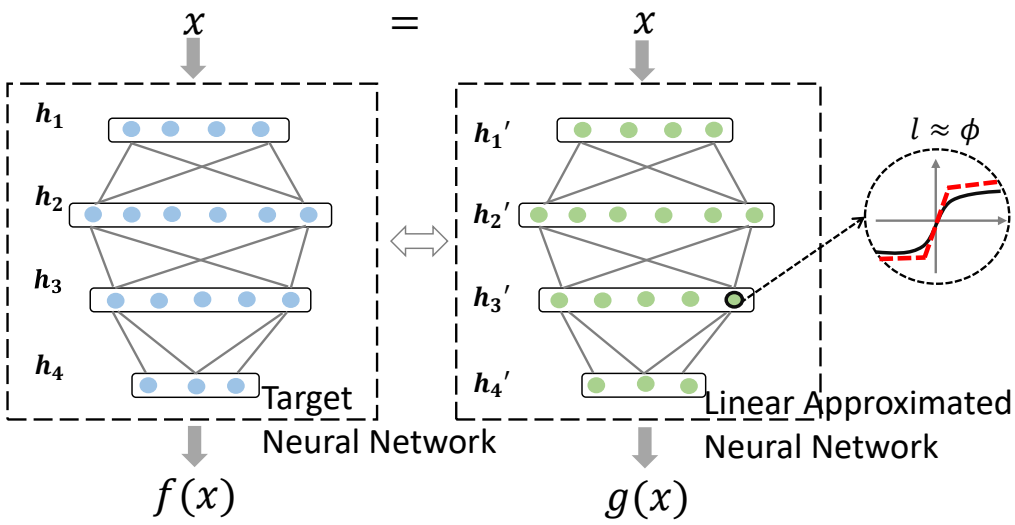


Figure 5.3: The structure of a LANN.

we use the number of linear regions in the approximation as a reflection of the complexity of the target model.

Given a target model that is a trained DNN with curve activation function, we propose *linear approximation neural network* (LANN), which is a piecewise linear approximation of the target model.

Definition 5 (Linear Approximation Neural Network)**.** *Given a fully connected neural network $f : \mathbb{R}^d \to \mathbb{R}^c$, a linear approximation neural network $g : \mathbb{R}^d \to \mathbb{R}^c$ is an approximation of f in which each activation function $\phi(\cdot)$ in f is replaced by a piecewise linear approximation function $\ell(\cdot)$.*

The LANN shares the same layer depth, width as well as weight matrix and bias vector with the target model, except that is approximates every activation function using an individual piecewise linear function. This brings two advantages. First, designing an individual approximation function for each neuron makes the approximation degree of a LANN model g to its target model f flexible and controllable. Second, the number of subfunctions of neurons is able to reflect the nonlinearity of the network. These two advantages will be further discussed in Section 5.3.2 and Section 5.3, respectively. A piecewise linear function

$\ell(\cdot)$ consisting of k subfunctions (linear regions), and can be written in the following form.

$$\ell(z) = \begin{cases} \alpha_1 z + \beta_1, & \text{if} \quad \eta_0 < z \leq \eta_1 \\ \alpha_2 z + \beta_2, & \text{if} \quad \eta_1 < z \leq \eta_2 \\ \vdots & \\ \alpha_k z + \beta_k, & \text{if} \quad \eta_{k-1} < z \leq \eta_k \end{cases} \tag{5.3}$$

where $\alpha_i, \beta_i \in \mathbb{R}$ are the parameters of the i-th subfunction. Given a variable z, the i-th subfunction is **activated** if $z \in (\eta_{i-1}, \eta_i]$, denote by $s(z) = i$. Let $\alpha^* = \alpha_{s(z)}$ and $\beta^* = \beta_{s(z)}$ be the parameters of the activated subfunction. We have $\ell(z) = \alpha^* z + \beta^*$.

Let $\phi_{i,j}$ be the activation function of the neuron $\{i, j\}$, where i is the index of the layer and j the id of the neuron in the layer. Then, $\ell_{i,j}$ is the approximation of $\phi_{i,j}$. Let $\ell_i = \{\ell_{i,1}, \ell_{i,2}, \ldots, \ell_{i,m_i}\}$ be the set of approximation functions for i-th hidden layer, m_i is the width of i-th hidden layer. The i-th layer of a LANN has the following form.

$$h'_i(z) = \ell_i(V_i z + b_i) \tag{5.4}$$

A LANN is in the form of

$$g(x) = V_o h'_L(h'_{L-1}(\ldots(h'_1(x)))) + b_o \tag{5.5}$$

Since the composition of piecewise linear functions is piecewise linear, a LANN is a piecewise linear neural network. Denote by $s_{i,j}$ the activation status of neuron $\{i, j\}$. We follow the convention in [104] and call the collection of statuses of all neurons the **activation pattern**[1].

Definition 6 (Activation pattern)**.** *An activation pattern of a piecewise linear neural network is the set of activation statuses of all neurons, denoted by* $s = \{s_{1,1}, \ldots, s_{1,m_1}, \ldots, s_{L,1}, \ldots, s_{L,m_L}\}$*, where* $s_{i,j}$ *is the activation status of neuron* $\{i, j\}$*.*

Given an input instance x, the corresponding activation pattern $s(x)$ is determined. The transformation of ℓ_i of layer i is reduced to a linear transformation that can be written in

[1]The activation pattern in this section and the configuration in Section 3 are with similar meaning. We use the term "activation pattern" to align with related works [104] in model complexity.

the following square matrix.

$$L_i = \begin{bmatrix} \alpha^*_{i,1} & 0 & \dots & 0 & \beta^*_{i,1} \\ 0 & \alpha^*_{i,2} & \dots & 0 & \beta^*_{i,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \alpha^*_{i,m_i} & \beta^*_{i,m_i} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \tag{5.6}$$

where $\alpha^*_{i,j}$ and $\beta^*_{i,j}$ are the parameters of the activated subfunction of neuron $\{i, j\}$, which is determined by $s_{i,j}$. The piecewise linear neural network is reduced to a linear function $y = Wx + b$ with the weight and bias value

$$\begin{bmatrix} W & b \end{bmatrix} = \begin{bmatrix} V_o & b_o \end{bmatrix} \prod_{i=L,\dots,1} \left(L_i \begin{bmatrix} V_i & b_i \\ \mathbf{0} & 1 \end{bmatrix} \right) \tag{5.7}$$

Obviously, an activation pattern corresponds to a linear region of the piecewise linear neural network. Given two different activation patterns, the square matrix L_i of at least one layer is different, so is the corresponding linear functions. Thus, the linear regions of a piecewise linear neural network can be represented by activation patterns. We define the linear region that includes input instance x by the activation pattern $s(x)$.

5.3.2 Degree of Approximation

We measure the complexity of models with respect to approximation degrees. Thus, we first provide a measure of approximation degree using approximation error. Then, we analyze approximation error of LANN in terms of neuronal approximation functions.

Definition 7 (Approximation error)**.** *Let $f' : \mathbb{R} \to \mathbb{R}$ be an approximation function of $f : \mathbb{R} \to \mathbb{R}$. Given input x, the approximation error of f' at x is $e(x) = |f'(x) - f(x)|$.*

where $|\cdot|$ denotes the absolute value. This definition describes the approximation error at a specific input. Below we define approximation error at model level which reflects the general degree of approximation of a model:

Definition 8 (Approximation error of LANN)**.** *Given a deep neural network $f : \mathbb{R}^d \to \mathbb{R}^c$ and a linear approximation neural network $g : \mathbb{R}^d \to \mathbb{R}^c$ learned from f. We define the approximation error of g to f as the expectation of the absolute distance between their outputs:*

$$\mathcal{E}(g; f) = \mathbb{E}[\ \frac{1}{c} \sum |g(x) - f(x)|\ ] \tag{5.8}$$

Since a LANN is learned by conducting piecewise linear approximation to every activation function, the approximation of every activation contributes some approximation error. The approximation error of a LANN is the accumulation of the approximation error on every neuron.

In literature [33, 104], approximation error of activation is treated as small perturbation added to a neuron, then, the exponential growth of neuron perturbations through multiple layers during forward propagation is discussed. Based on this, we go one step further to estimate the contribution of perturbation of every neuron to the model prediction through error propagation analysis.

Consider a target model f and its LANN approximation g. According to Definition 7, the approximation error of $\ell_{i,j}$ of g corresponding to neuron $\{i, j\}$ can be rewritten to $e_{i,j} = |\ell_{i,j} - \phi_{i,j}|$.

Suppose the same input instance is fed into f and g simultaneously. After the forward computation of the first i hidden layers, the output of the i-th layer of f and g are different. Let r_i be the output difference of the i-th hidden layer between g and f, and r_{i-1} for the $(i-1)$-th layer. Let x be the input to the i-th layer as well as the output of the $(i-1)$-th layer of f. We can compute r_i by

$$r_i = h'_i(x + r_{i-1}) - h_i(x) \tag{5.9}$$

The absolute value of r_i is

$$\begin{aligned} |r_i| &= |h'_i(x + r_{i-1}) - h_i(x)| \\ &= |h'_i(x + r_{i-1}) - h_i(x + r_{i-1}) + h_i(x + r_{i-1}) - h_i(x)| \\ &\leq |h'_i(x + r_{i-1}) - h_i(x + r_{i-1})| + |h_i(x + r_{i-1}) - h_i(x)| \end{aligned} \tag{5.10}$$

To keep the discussion simple, we write $x_r = x + r_{i-1}$. The first term of the righthand side of Eq. (5.10) is

$$|h'_i(x_r) - h_i(x_r)| = e_i(V_i x_r + b_i), \tag{5.11}$$

where $e_i = [e_{i,1}, e_{i,2}, \ldots, e_{i,m_i}]^T$ is a vector consisting of every neuron's approximation error of the i-th layer. Applying the first-order Taylor expansion to the second term of Eq. (5.10), we have:

$$|h_i(x + r_{i-1}) - h_i(x)| = |J_i(x) r_{i-1} + \epsilon_i| \tag{5.12}$$

where $J_i(x) = \frac{dh_i(x)}{dx}$ is the Jacobian matrix of the i-th hidden layer of f, ϵ_i is the remainder of the first-order Taylor approximation . Plugging Eq. (5.11) and Eq. (5.12) into Eq. (5.10),

we have:

$$|r_i| \leq e_i(V_i x_r + b_i) + |J_i(x) r_{i-1}| + |\epsilon_i| \tag{5.13}$$

Assuming x and r_{i-1} being independent, the expectation of $|r_i|$ is

$$\mathbb{E}[|r_i|] \leq \mathbb{E}[e_i] + \mathbb{E}[|J_i|]\ \mathbb{E}[|r_{i-1}|] + \mathbb{E}[\hat{\epsilon_i}] \tag{5.14}$$

where the error $\hat{\epsilon_i} = \epsilon_i + \varepsilon_i$, ε_i denotes the error in $\mathbb{E}[e_i]$, in other words, the disturbances of r_{i-1} on the distribution of e_i. Since $\mathbb{E}[e_i]$ is a vector where the elements correspond to the neurons in the i-layer layer, the expectation of $e_{i,j}$ is:

$$\mathbb{E}[e_{i,j}] = \int e_{i,j}(x) t_{i,j}(x) dx, \tag{5.15}$$

where $t_{i,j}(x)$ is probability density function (PDF) of neuron $\{i, j\}$.

We notice that $h_i(x)$ consists of a linear transformation $V_i x + b_i$ followed by activation ϕ. So the Jacobian matrix can be computed by $J_i(x) = \phi' \circ V_i$. The j-th row of $E[|J_i|]$ is

$$\mathbb{E}[|J_i|]_{j,*} = \int |\phi'(x)| t_{i,j}(x) dx \circ |V_i|_{j,*} \tag{5.16}$$

where the subscript $j, *$ means the j-th row of the matrix.

The above process describes the propagation of approximation error through the i-th hidden layer. Applying the propagation calculation recursively from the first hidden layer to the output layer, we have the following result.

Theorem 3 (Approximation error propagation)**.** *Given a deep neural network $f : \mathbb{R}^d \to \mathbb{R}^c$ and a linear approximation neural network $g : \mathbb{R}^d \to \mathbb{R}^c$ learned from f. The approximation error*

$$\mathcal{E}(g; f) = \frac{1}{c} \sum (|V_o|\ \mathbb{E}[|r_L|]), \tag{5.17}$$

where, for $i = 2, \ldots, L$,

$$\mathbb{E}[|r_i|] \leq \mathbb{E}[e_i] + \mathbb{E}[|J_i|]\mathbb{E}[|r_{i-1}|] + \mathbb{E}[|\hat{\epsilon_i}|] \tag{5.18}$$

and $\mathbb{E}[|r_1|] = \mathbb{E}[e_1]$.

Based on Theorem 3, expanding Eq. (5.18), we have

$$\mathbb{E}(|r_L|) \approx \sum_{i=1}^{L} \prod_{q=L}^{i+1} \mathbb{E}[|J_q|](\mathbb{E}[e_i] + \mathbb{E}[|\hat{\epsilon}_i|]) \tag{5.19}$$

Plugging Eq. (5.19) into Eq. (5.17), the model approximation error $\mathcal{E}(g; f)$ can be rewritten in terms of $E[e_{i,j}]$, that is,

$$\mathcal{E}(g; f) = \sum_{i,j} \underbrace{\frac{1}{c} \sum(\ |V_o| \prod_{q=L}^{i+1} \mathbb{E}[|J_q|] \ )_{*,j}}_{w_{i,j}^{(e)}} (\mathbb{E}[e_{i,j}] + \mathbb{E}[|\hat{\epsilon}_{i,j}|]) \tag{5.20}$$

here $\sum(\cdot)_{*,j}$ sums up the j-th columns. $w_{i,j}^{(e)}$ is the amplification coefficient of $\mathbb{E}[e_{i,j}]$ reflecting its amplication in the subsequent layers to influence the output. $w_{i,j}^{(e)}$ is independent from the approximation of g and is only determined by f. When $\mathcal{E}(g; f)$ is small and the approximation of g is very close to f, the error $\hat{\epsilon}_i$ can be ignored, $\mathcal{E}(g; f)$ is roughly considered a linear combination of $\mathbb{E}[e_{i,j}]$ with amplification coefficient $w_{i,j}^{(e)}$.

5.3.3 Approximation Algorithm

We use the smallest LANN that meets the minimum degree of approximation, which measured by approximation error $\mathcal{E}(g; f)$, to assess the complexity of a model. Given a threshold of approximation degree, we minimize the number of linear regions needed by the approximation. The number of linear regions reflects how nonlinear the function of the target model is. Unfortunately, the actual number of linear regions corresponding to data manifold [10] in the input-space is unknown. To tackle the challenge, we notice that a piecewise linear activation function with k subfunctions contributes $k - 1$ hyperplanes to the input-space partition [22, 91]. To minimize the number of linear regions, we minimize the number of hyperplanes of input-space partition. Formally, under a requirement of approximation degree λ, our algorithm learns a LANN model with minimum $K(g) = \sum_{i,j} k_{i,j}$.

$$\min(K(g)) \quad s.t. \ \ \mathcal{E}(g; f) < \lambda \tag{5.21}$$

Before presenting our algorithm, we first introduce how we obtain the neuron distribution $t_{i,j}$.

Distribution of activation function

In Section 5.3.2, in order to compute $\mathbb{E}[e_{i,j}]$ and $\mathbb{E}[|J_i|]$, we introduce the probability density function $t_{i,j}$ of the activation of neuron $\{i,j\}$. To compute $t_{i,j}$, the distribution of activation function is involved. The distribution of an activation function is how outputs (or inputs) of a neuronal activation function distribute with respect to the data manifold. It is influenced by the parameters of previous layer and the distribution of input data. Since the common curve activation functions are bounded to a small output range, to simplify the calculation, we study the posterior distribution of an activation function [38, 64] instead of the input distribution. To estimate the posterior distribution of neuron $\{i,j\}$, we use kernel density estimation (KDE) [113] with Gaussian kernel, and use the output of activation function $\phi_{i,j}$ on training dateset as the distributed samples $\{x_1, x_2, \ldots, x_n\}$.

$$t_{i,j} = \frac{1}{nh} \sum_{q=1}^{n} K(\frac{x - x_q}{h}) \tag{5.22}$$

where the bandwidth h is chosen by the rule-of-thumb estimator [113]. To compute $\mathbb{E}[e_{i,j}]$ and $\mathbb{E}[|J_i|]$, we uniformly sample n_t points $\{\Delta x_1, \ldots, \Delta x_{n_t}\}$ within the output range of ϕ, where distance between two neighbor samples is $\Delta x = \frac{\phi(\infty) - \phi(-\infty)}{n_t}$. We then use the expectation on these samples as an estimation of $\mathbb{E}[e_{i,j}]$.

$$\mathbb{E}[e_{i,j}] \approx \sum_{q=1}^{n_t} e_{i,j}(\Delta x_q) t_{i,j}(\Delta x_q) \tag{5.23}$$

The output of ϕ is smooth and in small range, setting large sample size n_t will not bring obvious improvement in the expectation estimation. In our experiments, we set $n_t = 200$. Notice that x_q is the output of ϕ. The corresponding input is $\phi^{-1}(x_q)$, so $e_{i,j}(x_q) = |\ell_{i,j}(\phi^{-1}(\Delta x_q)) - \Delta x_q|$. $\mathbb{E}[|J_i|]$ is computed in the same way.

Piecewise linear approximation of activation

To minimize $K(g)$, the piecewise linear approximation function $\ell_{i,j}$ of an arbitrary neuron $\{i,j\}$ is initialized with a linear function ($k = 1$). Then every new subfunction is added to $\ell_{i,j}$ to minimize the value of $\mathbb{E}[e_{i,j}]$. Every subfunction is a tangent line of ϕ. The initialization is the tangent line at $(0, \phi(0))$, which corresponds to the linear regime of activation function [64]. A new subfunction is added to the *next tangent point* $(p^*, \phi(p^*))$, which is

Algorithm 3 nextTangentPoint

Input: ϕ, ℓ, t

Output: p^*, $\mathbb{E}[e]_-$

$\{\Delta x_1, \ldots, \Delta x_{n_t}\} \leftarrow$ uniformly sampled points

//Compute $\mathbb{E}[e]$ by Eq. (5.23)

for Δx **in** $\{\Delta x_1, \ldots, \Delta x_{n_t}\}$ **do**

  $\ell'_{\Delta x}$ = add tangent line of Δx to ℓ

  Compute $\mathbb{E}[e(\ell'_{\Delta x})]$

end for

$\Delta x^* = \arg\min_{\Delta x} \mathbb{E}[e(\ell'_{\Delta x})]$

$\mathbb{E}[e]_- = \mathbb{E}[e] - \mathbb{E}[e(\ell'_{\Delta x*})]$

found from the set of uniformly sampled points $\{\Delta x_1, \Delta x_2, \ldots, \Delta x_{n_t}\}$. That is,

$$p^*_{i,j} = arg\min_p \mathbb{E}[e_{i,j}]_{+p}; \quad p \in \{\Delta x_1, \ldots, \Delta x_{n_t}\} \tag{5.24}$$

where subscript $+p$ means $\ell_{i,j}$ with additional tangent line of $(p, \phi(p))$ is used in computing $\mathbb{E}[e_{i,j}]$. Algorithm 3 shows the pseudocode of determining the next tangent point.

Algorithm of building LANNs

To minimize $K(g)$, the algorithm starts with initializing every approximation function $\ell_{i,j}$ with $k = 1$. g is a linear function. Then, we iteratively add a subfunction to the approximation function of a certain neuron to decrease $\mathcal{E}(g; f)$ to the most degree in each step.

Considering Eq. (5.20), when building a LANN the error $\hat{\epsilon}_i$ cannot be ignored because $\mathbb{E}[e_{i,j}]$ is large. The amplification coefficient $w^{(e)}_{i,j}$ of lower layer is exponentially larger than that of upper layer. Otherwise, error $\mathbb{E}[\hat{\epsilon}_{i,j}]$ grows exponentially from lower to upper layer. Deriving this formula to get the exact weight of $\mathbb{E}[e_{i,j}]$ is complicated. A simple way is to roughly consider each $\mathbb{E}[e_{i,j}]$ to be equally important in the algorithm. For a neuron from the first layer, small $\mathbb{E}[e_{i,j}]$ is desired due to large magnitude of $w^{(e)}_{i,j}$ even through $\mathbb{E}[\hat{\epsilon}_{i,j}] = 0$. Another neuron from the last hidden layer, its amplification coefficient $w^{(e)}_{i,j}$ is with lowest magnitude over all layers but the $\mathbb{E}[\hat{\epsilon}_{i,j}]$ is unignorable and might influence the distribution of neuron status, thus approximation with small $\mathbb{E}[e_{i,j}]$ is desired to decrease the value of $\mathbb{E}[e_{i,j}]$ and $\mathbb{E}[\hat{\epsilon}_{i,j}]$.

Algorithm 4 outlines the LANN building algorithm. Below we describe the main steps. To reduce the calculation times of $\mathcal{E}(g; f)$, we set up the batchsize b to batch processing a group of neurons.

Algorithm 4 BuildingLANN

Input: a DNN $f(x)$ with activation function ϕ; training dataset D_{tr}; a set of activation function distributions $T = \{t_{i,j}\}$; batchsize b; approximation degree λ

Input: a LANN model g

Initialize $\ell_{i,j}$ in g with linear functions

$//w_{i,j}^{(e)} \leftarrow$ compute neuron weight (Eq. (5.20))

for $i \leftarrow 1$ to L **do**

  for $j \leftarrow 1$ to m_i **do**

    Compute $\mathbb{E}[e_{i,j}]$ by Eq.(5.23)

    $p_{i,j}^*,\ \mathbb{E}[e_{i,j}]_- \leftarrow$nextTangentPoint$(\phi, \ell_{i,j}, t_{i,j})$

  end for

end for

while $\mathcal{E}(g; f) \leq \lambda$ **do**

  $N_u =$ select b neurons with maximum $E[e_{i,j}]_-$

  for every neuron $u \in N_u$ **do**

    $\ell_u \leftarrow$ add tangent line of p_u^* to ℓ_u

    $\mathbb{E}[e_u] = \mathbb{E}[e_u] - \mathbb{E}[e_u]_-$

    $p_u^*, \mathbb{E}[e_u]_- \leftarrow$nextTangentPoint$(\phi, \ell_{i,j}, t_{i,j})$

  end for

  $\mathcal{E}(g; f) \leftarrow$ approximation error on D_{tr}

end while

a) Initialize g to set every $\ell_{i,j}$ to be a linear function ($k_{i,j} = 1$). Find next tangent point $p_{i,j}^*$ for every $\ell_{i,j}$.

b) Find neuron $\{i^*, j^*\}$

$$\{i^*, j^*\} = \arg\max(\mathbb{E}[e_{i,j}] - \mathbb{E}[e_{i,j}]_{+p_{i,j}^*}), \tag{5.25}$$

Tangent line of $(p_{i^*,j^*}^*, \phi(p_{i^*,j^*}^*))$ is considered currently the most effective addition.

c) Add the new subfunction to ℓ_{i^*,j^*}. Then update its next tangent point p_{i^*,j^*}^*.

d) Repeat Steps b) and c) until $\mathcal{E}(g; f) < \lambda$.

The complexity of the algorithm (Algorithm 4) is $O(K(g)n)$. The time cost of the first loop is $O((\sum_{i=1}^{L} m_i) * n_t^2)$. The second loop repeats $(K(g) - \sum_{i=1}^{L} m_i)$ times, within each loop the computation cost is $O((\sum_{i=1}^{L} m_i) + n_t^2 + n)$, where n_t is the number of segment of $t(x)$, n is the number of instances of D_{tr}.

5.4 Model Complexity

The previous section described how to build a linear approximation neural network to approximate a target model so that the approximation satisfies a given approximation degree requirement and has as few linear regions as possible. The number of linear regions reflects how nonlinear the classification function of the target model is, which is complexity of the target model. In this section, we propose a tight upper bound to the number of linear regions of a LANN, then define a complexity measure based on the upper bound.

The idea of measuring model complexity using the number of linear regions is popular in piecewise linear neural networks [91, 97, 100, 104]. Montufar *et al.* [91] and Raghu *et al.* [104] presented an upper bound to the number of linear regions in piecewise linear neural networks. We generalize their results to the LANN model, of which the major difference is that, in LANN, each piecewise linear activation function has different forms and different numbers of subfunctions. We propose a tight upper bound to the number of linear regions formed by a LANN model.

Theorem 4 (Upper bound)**.** *Given a linear approximation neural network $g : \mathbb{R}^d \to \mathbb{R}^c$ with L hidden layers. Let m_i be the width of the i-th layer and $k_{i,j}$ the number of subfunctions of $\ell_{i,j}$. The number of linear regions of g is upper bounded by*

$$\prod_{i=1}^{L}(\sum_{j=1}^{m_i} k_{i,j} - m_i + 1)^d. \tag{5.26}$$

Proof. First of all, according to [91, 100, 104], the total number of linear regions divided by k hyperplanes in the input space $\mathbb{R}^d$ is upper bounded by $\sum_{i=0}^{d} \binom{k}{i}$, whose upper bound can be obtained using binomial theorem:

$$\sum_{i=0}^{d} \binom{k}{i} \leq (k+1)^d. \tag{5.27}$$

Now consider the first hidden layer h'_1 of a LANN model. A piecewise linear function consisting of $k_{i,j}$ subfunctions contributes $k_{i,j} - 1$ hyperplanes to the input space splitting. The first layer h'_1 contains m_1 neurons, with j-th neuron consisting of $k_{1,j}$ subfunctions. So h'_1 contributes $\sum_{j=1}^{m_1}(k_{1,j} - 1)$ hyperplanes to the input space $\mathbb{R}^d$ splitting, and divides $\mathbb{R}^d$ into at most

$$(\sum_{j=1}^{m_1} k_{1,j} - m_1 + 1)^d \tag{5.28}$$

linear regions.

Now move to the second hidden layer h'_2. For each linear region divided by the first layer, it can be divided by the hyperplanes of h'_2 to at most $(\sum_{j=1}^{m_2} k_{2,j} - m_2 + 1)^d$ smaller regions.

Thus, the total number of linear regions igenerated by h'_1, h'_2 is at most

$$(\sum_{j=1}^{m_1} k_{1,j} - m_1 + 1)^d * (\sum_{j=1}^{m_2} k_{2,j} - m_2 + 1)^d. \tag{5.29}$$

Recursively do this calculation until the last hidden layer h'_L. The number of linear regions divided by g is at most

$$\prod_{i=1}^{L} (\sum_{j=1}^{m_i} k_{i,j} - m_i + 1)^d. \tag{5.30}$$

□

This theorem indicates that the number of linear regions is polynomial with respect to layer width and exponential with respect to layer depth. This is consistent with the previous studies on the power of neural networks [8, 9, 33, 83, 102, 104]. Meanwhile, the value of k reflects the nonlinearity of neurons according to the status distribution of activation functions. The distribution is influenced by both model parameters and input data manifold. So this upper bound to some degree reflects the impact of model parameters. Based on this upper bound, we define a complexity measure:

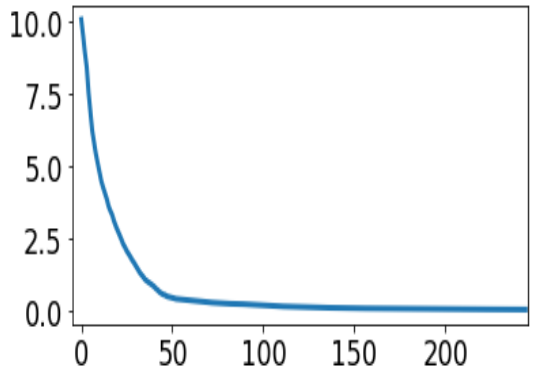
Definition 9 (Complexity measure)**.** *Given a deep neural network f and a linear approximation neural network g learned from f with approximation degree λ, the λ-approximation complexity measure of f is*

$$C(f)_\lambda = d \sum_{i=1}^{L} \log(\sum_{j=1}^{m_i} k_{i,j} - m_i + 1) \tag{5.31}$$

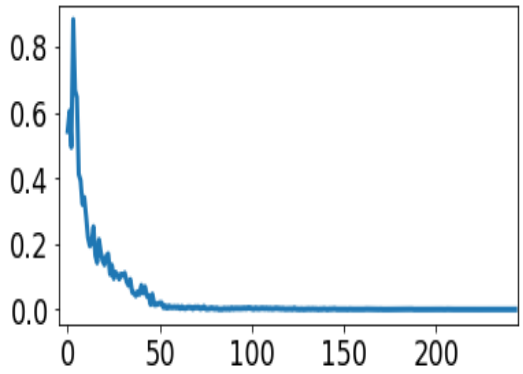
This complexity measure is essentially a simplification of our proposed upper bound by logarithm. In the following, we introduce our recommended method of setting the value of λ. To be briefly, our suggested λ located at the range when $(\mathcal{E}')^2/\mathcal{E}''$ converges to a constant.

5.4.1 Determine λ

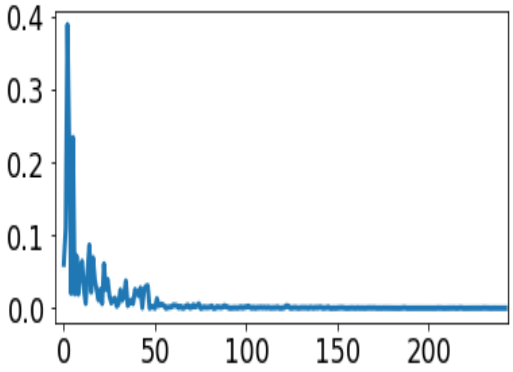
In this section, we suggest the range of λ when using LANN for complexity measure. A suitable value of λ makes the complexity measure trustworthy and stable. When the value



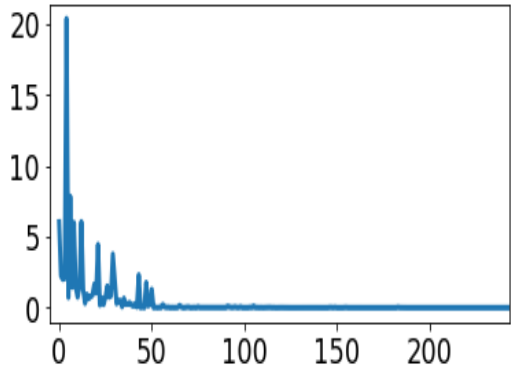
(a) Approximation error $\mathcal{E}$



(b) Approximation gain k



(c) Second-order derivative a

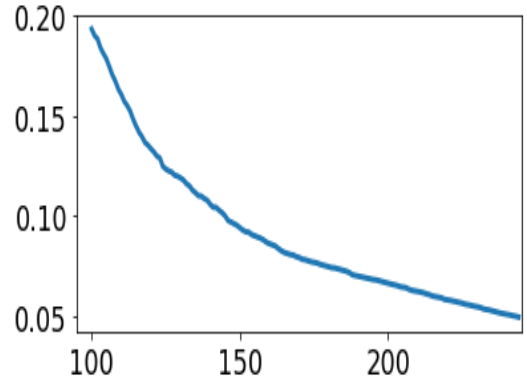


(d) k^2/a

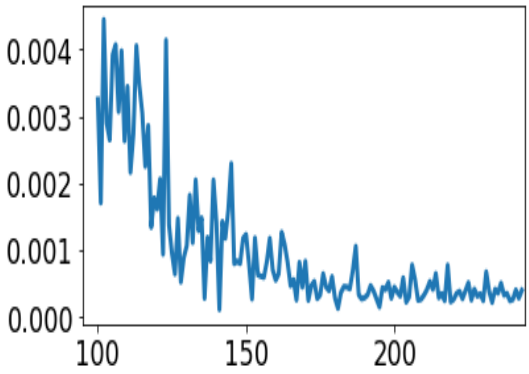
Figure 5.4: Changing trend of approximation error $\mathcal{E}$, approximation gain k, a which is the second-order derivative of $\mathcal{E}$, and k^2/a computed from k and a.

of λ is too large, the measure may be unstable and unable to reflect the real complexity. It seems a small value of λ is preferred. However, a small value calls for a higher cost to construct the LANN approximation. Based on analyzing the curve of the approximation error, we provide an empirical recommendation for the value range.

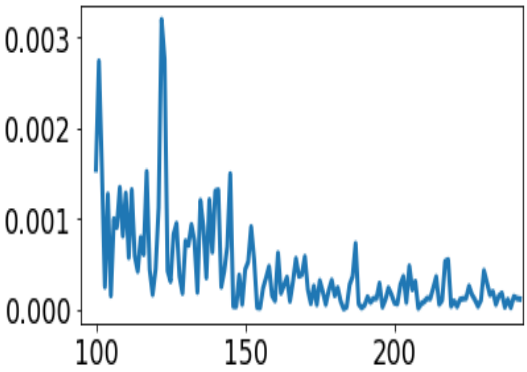
We first analyze the curve of approximation error in several aspects. The approximation error $\mathcal{E}$ is the optimization objective of the algorithm for building LANN (Algorithm 4), obviously the value of $\mathcal{E}$ will continue to decrease during building a LANN (Figure 5.4(a)). We name the first-order derivative of $\mathcal{E}$ as approximation gain, denoted by k. The approximation gain represents the contribution of the current epoch to the decrease of the approximation error $\mathcal{E}$. Our algorithm ensures that at any time during building a LANN, the value of k is larger than all remaining possible operations, that is, all later possible operations. Figure 5.4(b) shows the curve of approximation gain. Since we ignore the error



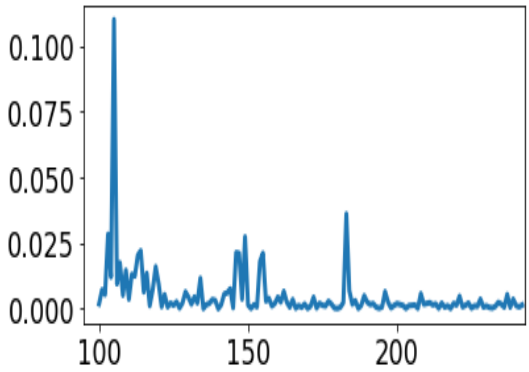
(a) Approximation error $\mathcal{E}$



(b) Approximation gain k



(c) Second-order derivative a



(d) k^2/a

Figure 5.5: Changing trend of approximation error $\mathcal{E}$, approximation gain k, a which is the second-order derivative of $\mathcal{E}$, and k^2/a computed from k and a. Here we enlarge second half, after 100 epoches of Figure 5.4.

$\hat{\epsilon}$ in the algorithm, the curve of approximation gain in practice has a small range of jitter. However, the continued downward trend of the approximation gain is obvious.

We also consider the derivative of k, which is the second-order derivative of the approximation error $\mathcal{E}$, denoted by a. a reflects the changing trend of k. It is easy to prove that, the trend of a goes decrease during building a LANN: if not, after a finite number of epochs we have $k = 0$. But in fact, $\mathcal{E}$ will never be reduced to zero, the operations in each epoch will bring a non-zero effect to $\mathcal{E}$, thus k will not be 0. Figure 5.4(c) shows the changing trend of a.

From Figure 5.4, it can be seen that the changing trends of the approximation error $\mathcal{E}$, the approximation gain k and the second-order derivative a are close to each other. The trend of the curve decreases rapidly at the beginning, then gradually tends to converge.

This is consistent with our algorithm design. After $\mathcal{E}$ flattens, the following relationships will be established: $k \to 0, a \to 0$, $k \neq 0, a \neq 0$, $a << k$.

Suppose there is a timestamp t_0 in the flatten region of $\mathcal{E}$, k, a represent the first-order, second-order derivative of $\mathcal{E}$. We show changing trends of the flatten regions in Figure 5.5. According to Figure 5.4 and the above analysis, the curve after t_0 is basically stable. We estimate the total gain of approximation error that can be brought by the remaining epochs. Suppose there exists a n that after n epochs from t_0, k becomes zero. Then the gain of the remaining epochs is the gain of the next n epochs. Suppose a is a constant, $n = k/a$. The gain of the remaining epochs is estimated by $kn - an^2/2 = k^2/2a$.

We analyze k and a from the view of the remaining gain estimation. k and a continue to decrease during building a LANN. If k and a goes almost stable and with a very close decreasing trend, the estimation of the remaining gain of t_0 should be close to the estimation of epochs around t_0. Suppose the above consition is ture, we have: $k^2/a \approx (k+a)^2/(a+a') \Rightarrow k/a \approx a/a'$, where a' is the derivative of a. This is, the downward trend of k and a are basically similar, and $a' << a << k << 1$ is true.

As a result, the phenomenon that k^2/a of a timestamp almost equals to the calculated value of its neighbors demonstrates that, the derivative of k and a are similar. The gain of remaining epochs are expected to be relatively stable, that is, each afterward epoch will not bring much influence to the value of $\mathcal{E}$. In this case, the approximation error $\mathcal{E}$ is relatively stable.

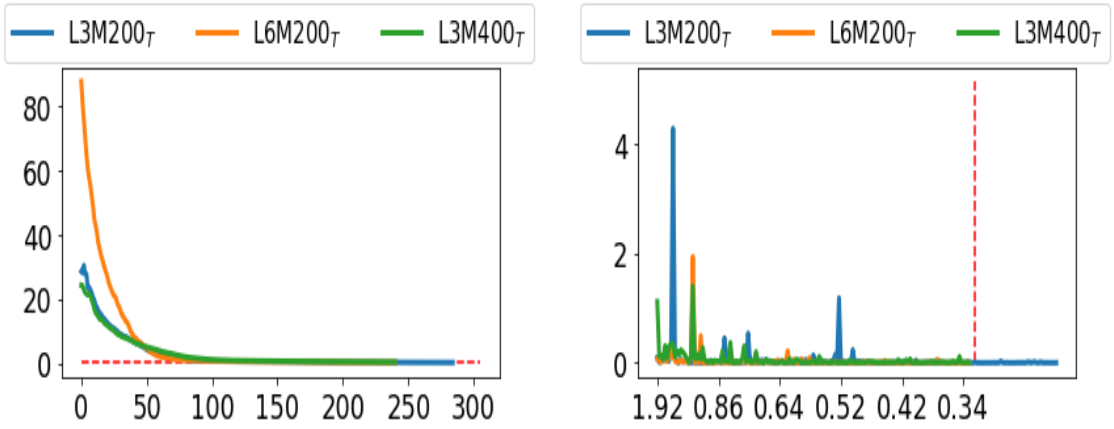
The conclusion is, for the construction of a LANN based on a specific target model, $\lambda < \lambda_0$ is suggested where λ_0 is the starting point of k^2/a converging to a constant.

When comparing two LANNs, the value of λ is $\lambda < min(\lambda_{0,a}, \lambda_{0,b})$ and $k_a(\lambda) \approx k_b(\lambda)$. This to some degree ensures the stability of complexity measure of the target models, since the estimated gain of remaining epochs of two LANNs are almost similar.

In practical experiments, the value of k^2/a is used to check whether the value of λ is reasonable. In our experiments, we choose a unified $\lambda = 0.1$ and verify its rationality. From our experimental results, it seems for a relatively simple network (e.g. 3 layers, hundreds of width), $\lambda \in [0, 0.12]$ is good enough since the k^2/a goes convergence. In Figure 5.6 we show why $\lambda = 0.1$ is a reasonable value in our experiments.

5.5 Insights from Model Complexity

In this section, we study several problems from the insights of complexity measure. First, we come out a directly obtained results of contributions of hidden neurons to model stability. We then investigate the changing trend of model complexity in training process. After that



(a) Approximation error $\mathcal{E}$

(b) k^2/a

Figure 5.6: Verification of the rationality of $\lambda = 0.1$ for three models trained on CIFAR: L3M200$_T$, L6M200$_T$, L3M400$_T$. The left figure shows the curve of approximation errors of three models. The right figure shows the value k^2/a in the area nearby 0.1. Here x axis is the corresponding approximation error.

	Sec 5.5.1	Sec 5.5.2	Sec 5.5.3, 5.5.4
MOON	-	-	L3M(32,128,16)$_T$
MNIST	L3M300$_S$	L3M100$_T$, L6M100$_T$, L3M200$_T$	-
CIFAR	L3M300$_T$	L3M200$_T$, L6M200$_T$, L3M400$_T$	L3M(768,256,128)$_T$

Table 5.1: Model structure of DNNs in our experiments.

we study the occurrence of overfitting and L^1, L^2 regularization. Finally, we propose two new simple and effective approaches for preventing overfitting.

Our experiments and evaluations are implemented on a synthetic dataset: the Two-Moons dataset [2], and two real-world datasets: the MNIST handwritten digits dataset [75] and the CIFAR-10 dataset [70]. To demonstrate the reliability of the complexity measure not depending on model structures, we design multiple model structures. We use $\lambda = 0.1$ for complexity measure in all experiments, who locates in our suggested range for all models we used. Table 5.1 summarizes the model structures we used. To read Table 5.1, L3 indicates the network is with 3 hidden layers, M300 means every layer contains 300 neurons while

[2]The synthetic dataset is generated by sklearn.datasets.make_moons API.

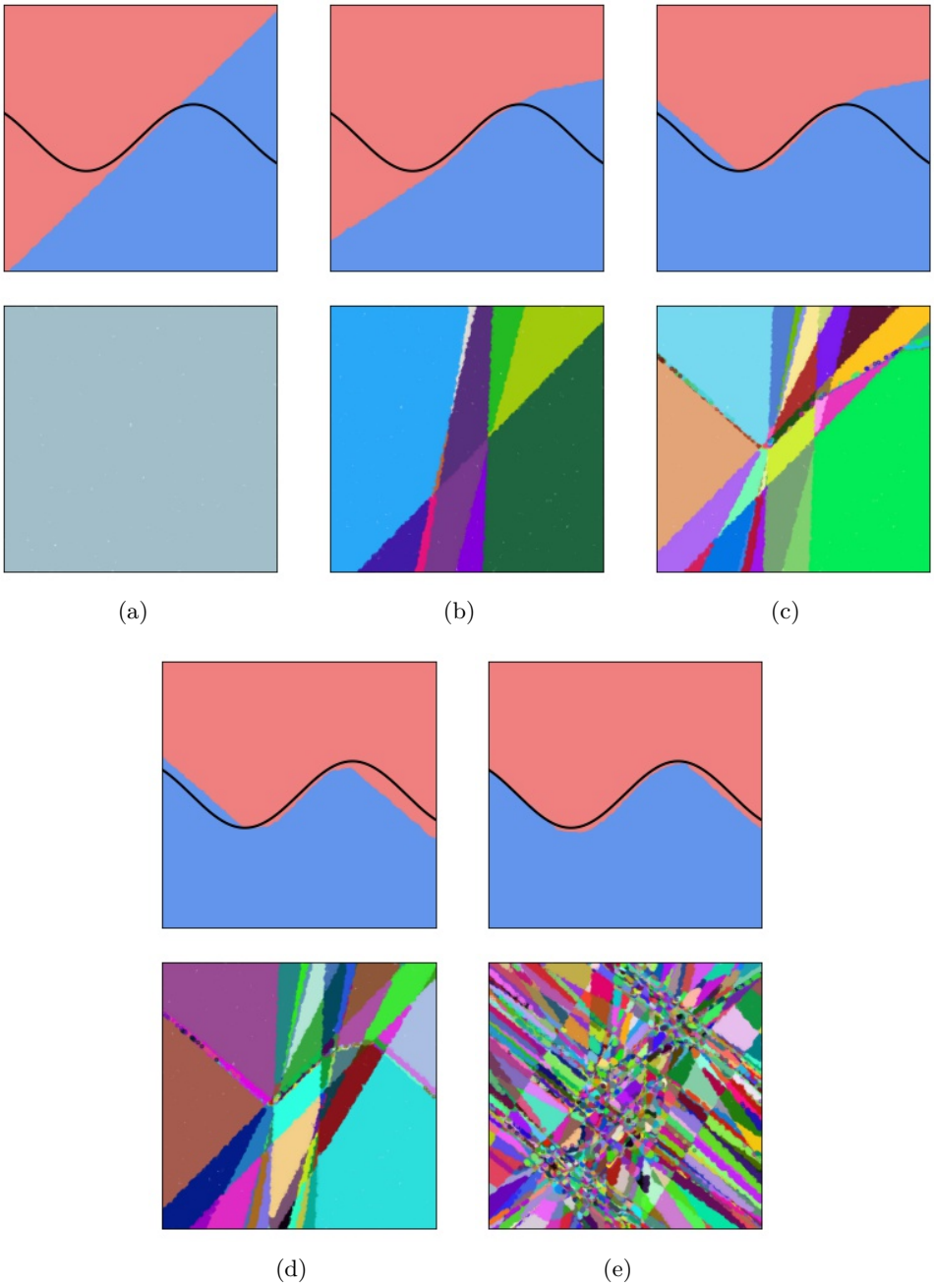
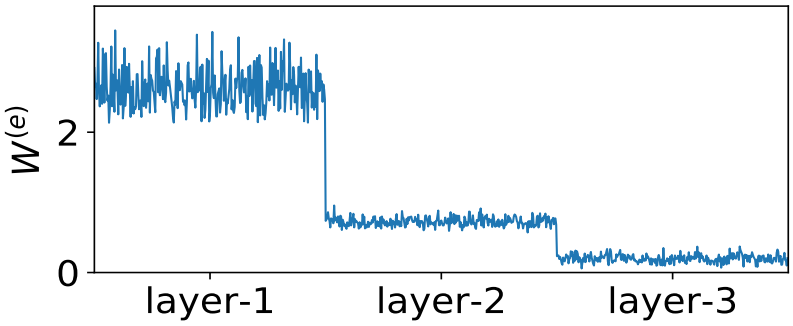
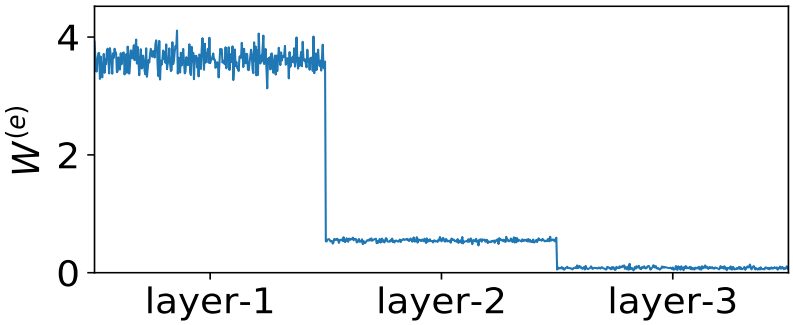


Figure 5.7: The evolution of decision boundary and linear regions when building a LANN, on the 'Sine' synthetic dataset. Each subfigure is an evolution moment during building a LANN. In each figure, the first row shows the classification boundary of the LANN comparing to the classification boundary of the deep model. Orange and blue correspond to different label. The second row shows the linear regions splitted by the LANN. One color corresponds to one linear region. Specifically, the 'Sine' is, $D_{syn} = \{(x, y)\}$ where $x = [x_1, x_2]^T, x_1, x_2 \in [-4, 4]$, $y = 1$ if $\sin(x_1) > x_2$ otherwise $y = 0$. The target binary classification network includes two hidden layer with 20 neurons in each layer and Sigmoid activation function.



(a) MNIST



(b) CIFAR

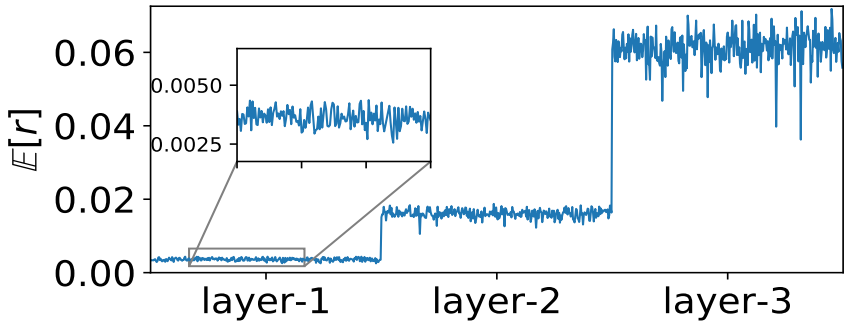
Figure 5.8: Amplification coefficient of every neuron.

M(32,128,16) means each layer contains 32, 128 and 16 neurons respectively. Subscripts S, T indicate the activation function type is Sigmoid or Tanh respectively.

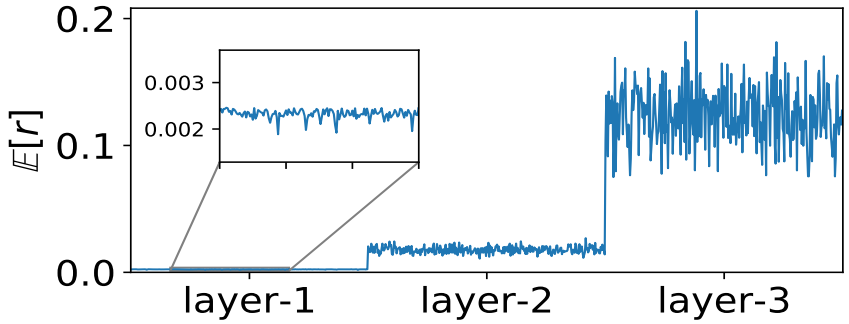
5.5.1 Hidden Neurons and Stability

The analysis in Section 5.3.2 gives out the amplification coefficient $w_{i,j}^{(e)}$ (Eq. 5.20), which measures how much a small purturbation on neuron $\{i, j\}$ is amplified during subsequent layers. In other words, the amplification coefficient reflects the effects of a neuron to model stability. The formulation of $w_{i,j}^{(e)}$ demonstrates the multiplication of $\mathbb{E}[J_p]$ through subsequent layer. Furthermore, our $w_{i,j}^{(e)}$ demonstrates the different contributions of neurons from the same layer. Figure 5.8 visualizes amplification coefficients of trained models on the MNIST, CIFAR datasets. To exclude the influence of layer width, each layer of the models is with equal width.

Besides amplification coefficient, we also visualize $\mathbb{E}[r_i]$, the error accumulation of all previous layers. According to our analysis, $\mathbb{E}[r_i]$ is expected to have opposite trend with

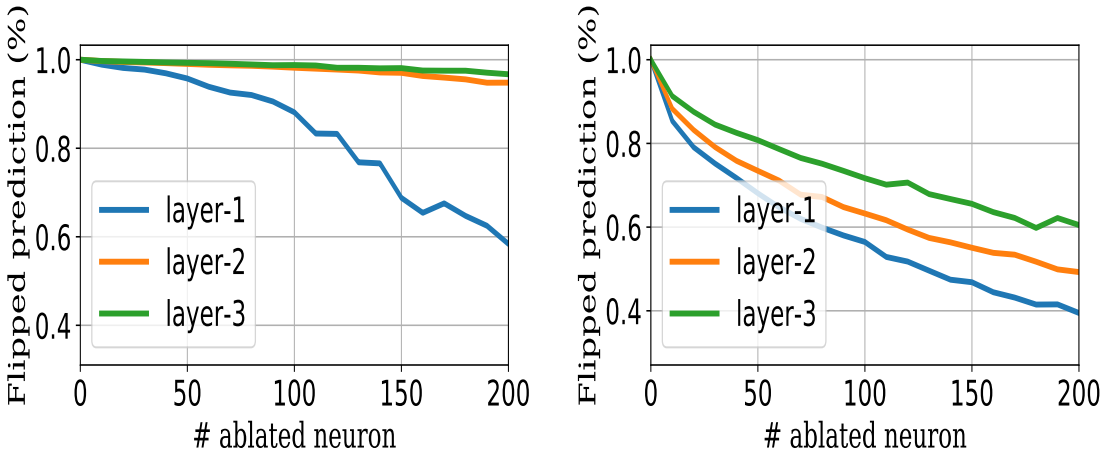


(a) MNIST



(b) CIFAR

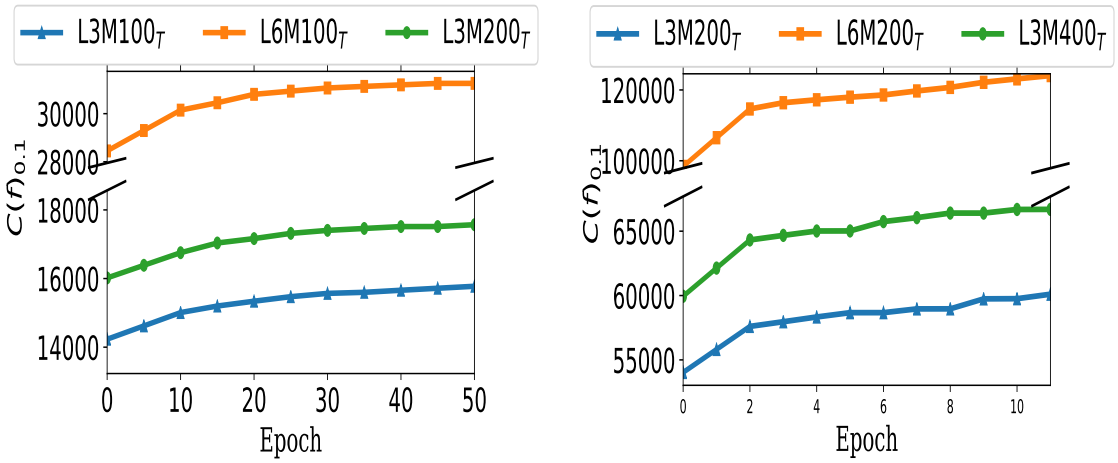
Figure 5.9: Layerwise error accumulation ($\lambda = 0.1$).



(a) MNIST

(b) CIFAR

Figure 5.10: Percentage of flipped prediction labels after random neuron ablation.



(a) MNIST

(b) CIFAR

Figure 5.11: Changing trend of complexity measure in training process of three models on MNIST dataset.

$w_{i,j}^{(e)}$: $\mathbb{E}[r]$ of upper layer is expected to be exponentially larger than lower layers. Figure 5.9 shows error accumulation $\mathbb{E}[r_i]$ on the same models.

To verify that a small perturbation on the lower layer influences the model outputs more than the upper layer, we randomly ablate neurons (fix neuron output to 0) from one layer of a well-trained model and observe the number of instances being flipped prediction labels. The results of ablating different layers are shown in Figure 5.10.

5.5.2 Complexity in Training

To investigate the changing trend of model complexity in the training process, we periodically measure the model complexity during training, after initialization. The 0.1-approximation complexity measure $C(f)_{0.1}$ are shown in Figure 5.11. Two important ideas come out of the results. First, model complexity keeps increasing during training. This indicates that the function of a deep model becomes more and more complicated. Second, comparing three structures tells us how the model structure influences the complexity measure.

The results demonstrate that increases in both width and depth can increase the model complexity. Further, with the same number of neurons, the complexity of a deep and narrow model (L6M100$_T$ on MNIST, L6M200$_T$ on CIFAR) is much higher than a shallow and wide one (L3M200$_T$ on MNIST, L3M400$_T$ on CIFAR). This agrees with existing studies on the effectiveness of width and depth of DNNs [8, 33, 91, 100].

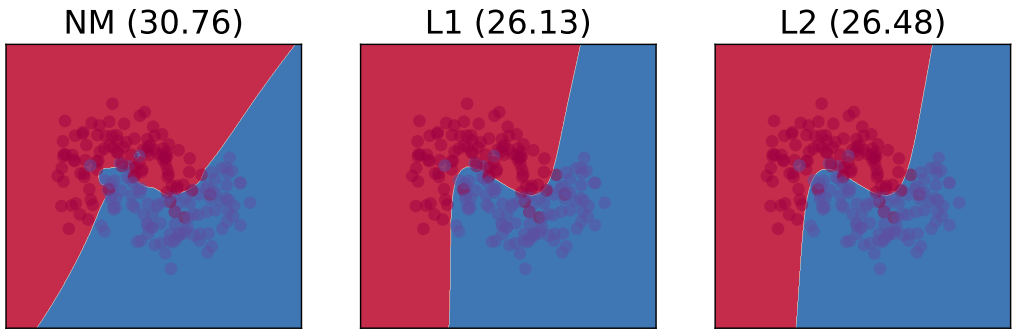


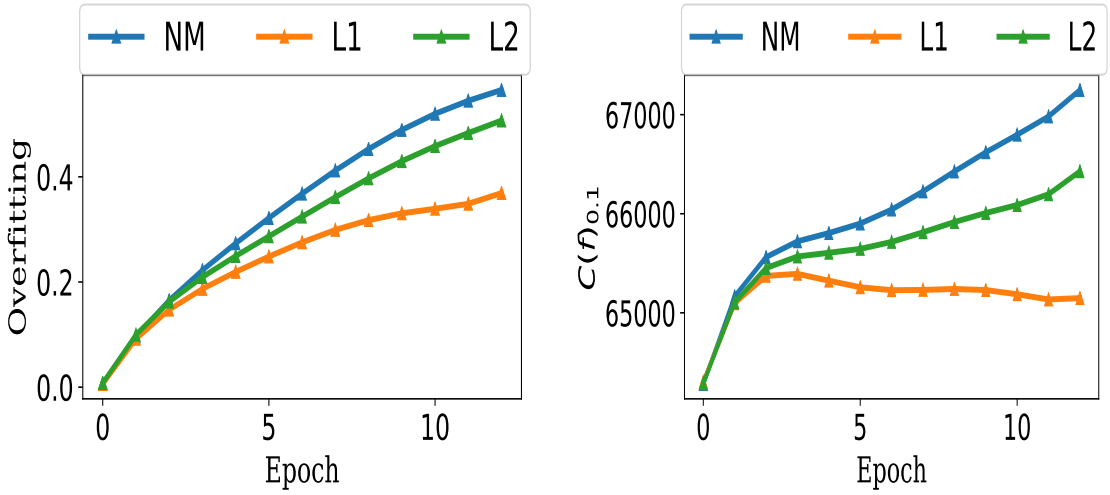
Figure 5.12: Decision boundaries of models trained on MOON dataset. NM, L1, L2 are short for normal train, train with L^1, L^2 regularization respectively. In brakets are the value of complexity measure $C(f)_{0.1}$.

5.5.3 Overfitting and Complexity

The complexity measured through LANNs can be used to understand overfitting. Overfitting occurs when using a model that is more flexible than is necessary [53]. Deep neural networks are able to accommodate curvilinear relationships and are highly flexible. As a result, the model maximizes its performance on the training set by memorizing data, rather than learning the patterns which can generalize to new data [45]. Discussion from previous studies points that an overfitted model will be more complex than another one that fits equally well [53]. This idea is also intuitively demonstrated by the polynomial fit example in Figure 5.1(b).

Regularization is an effective approach to prevent overfitting, by adding regularizer to the loss function, especially L^1 and L^2 regularization [45]. L^1 regularization results in a more sparse model, and L^2 regularization results in a model with small weight parameters. A guess is these regularization approaches which prevent overfitting are able to constrain the model complexity from increasing. To verify this, we train deep models on the MOON dataset with and without regularization. After 2,000 training epochs, their decision boundaries and complexity measure $C(f)_{0.1}$ are shown in Figure 5.12.

We also measure model complexity during the training process, after each epoch of CIFAR, with or without L^1, L^2 regularization. The results are shown in Figure 5.13. Figure 5.13(a) is the overfitting degree measured by $(Accuracy_{train} - Accuracy_{test})$, Figure 5.13(b) is the corresponding complexity measure $C(f)_{0.1}$. The results verify the conjecture that L^1, L^2 regularization constrain the model complexity increasing.



(a) Overfitting

(b) Complexity measure

Figure 5.13: Complexity measure during training of CIFAR dataset. Weight penalty are $1e-4$, $1e-3$ for L^1, L^2 regularization respectively.

	NM	PR	C-L1	L1	L2
$C(f)_{0.1}$	31.17	25.02	25.11	25.78	26.55
# Regions	45,772	182	356	382	545

Table 5.2: Complexity measure and number of linear regions of MOON.

5.5.4 New Approaches for Overfitting

Motivated by the finding that the increase of model complexity during training is significantly correlated with the occurrence of overfitting, we come out of two approaches to prevent overfitting, by directly constraining the increasing of model complexity.

First, **neuron pruning**. From the definition of complexity (Def. 9), we know that constraining model complexity $C(f)_\lambda$ is to constrain its variable $k_{i,j}$ for each neuron, which equals to constraining the nonlinearity of the distribution of a neuron. In this case, the approach we proposed is to periodically prune neurons with a maximum value of $\mathbb{E}[|t|]$, after each epoch's training. The reason is, closing to 0 is the linear range of a sigmoid, or tanh [58], with a larger value of $\mathbb{E}[t]$ the distribution t is with higher probability located at the nonlinear range and therefore requires larger k. Pruning neurons with a potentially large degree of nonlinearity can effectively constrain the increase of model complexity. At the

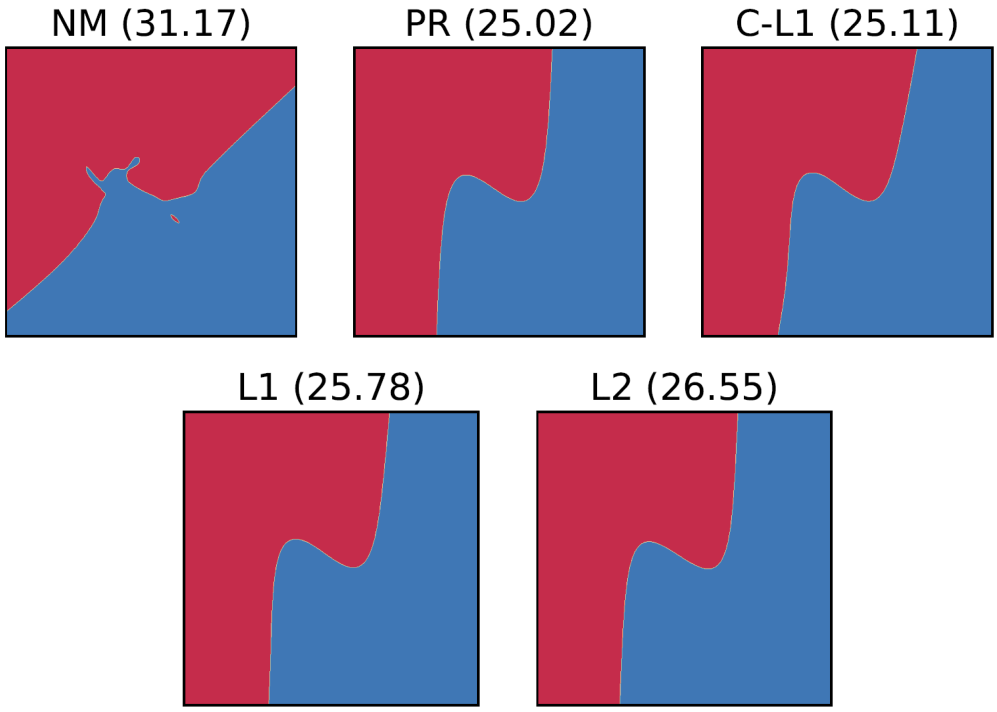
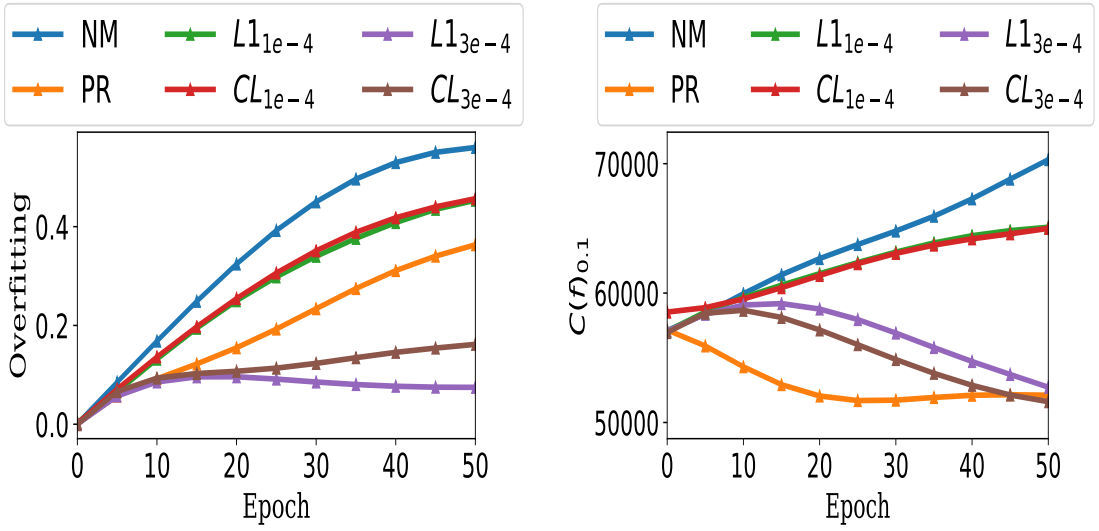


Figure 5.14: Decision boundaries of models trained with different regularization methods on MOON dataset. PR, C-L1 are short for training with neuron prunning, with customized L^1 regularization.



(a) Overfitting

(b) Complexity measure

Figure 5.15: Degree of overfitting and complexity measure in training process of CIFAR dataset.

same time, small number of prunning will not obviously decrease the model performance. Practical results demonstrate that this approach is simple, efficiency and really effective.

Second, **customized L^1 regularization**. This is to give customized coefficient to every column of weight matrix $V_i (i = 1, \ldots, L)$ when doing L^1 regularization. Each column corresponds to a specific neuron and with coefficient:

$$a_{i,j} = \mathbb{E}[|\phi'_{i,j}|] = \int |\phi'(x)| t_{i,j}(x) dx \tag{5.32}$$

One explanation is, $a_{i,j}$ equals to the expectation of first-order derivative of $\phi_{i,j}$. With larger value of $\mathbb{E}[|\phi'_{i,j}|]$, the distribution $t_{i,j}$ is with higher probability located at the linear range of the activation function ($0 = argmax_x \phi'(x)$). In this case, this customized L^1 approach assigns larger sparse penalty weight to more linearly distributed neurons. On the other hand, neurons with more nonlinear distribution can maintain their expressive power. Another view to understand this approach is by Eq. (5.19): $a_{i,j}|V_i| = \mathbb{E}[|J_i|]$. That is, the formulation of customized L^1 regularization can also be interpreted as the constraint of of $E[|J|]$, which will obviously result in smaller $\mathcal{E}(g; f)$ as well as smaller $C(f)$. Customized L^1 is more flexible than normal L^1 regularization, thus behaves better with large penalty weight.

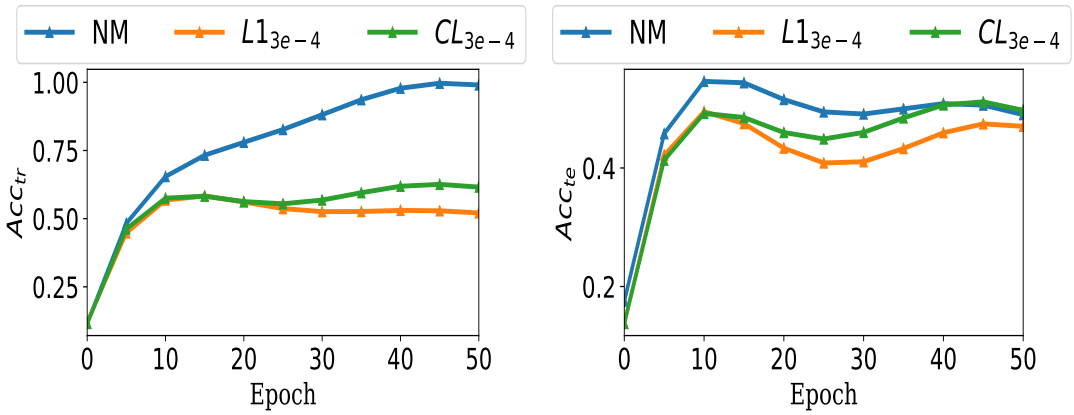


Figure 5.16: Left shows the accuracy on the CIFAR training dataset, the right one shows the accuracy on the CIFAR test dataset. Both in the training process.

In Figure 5.14 we comparing decision boundaries of models training on the MOON dataset, with different regularization approaches. Table 5.2 records the corresponding complexity measure and the account of number of linear regions the input space been split. In Figure 5.15 shows overfitting and complexity measure in the training process of a model on CIFAR. After each epoch 5% neurons are pruned. See from the results, neuron pruning maintain the performance on test dataset and shows effectiveness in constraining overfitting and complexity. Coefficients of customized L^1 are scaled such that the average equal to penalty weight of L^1. Results shows, with small penalty weight, customized L^1 behaves close to normal L^1. With large penalty weight, performance of L^1 model is affected, test accuracy decrease by 3%. Customized L^1 maintains the performance.

That is, the customized L^1 regularization is more flexible than normal L^1 regularization, and behaves better with large weight penalty. We report the experimental results that customized L^1 maintains the prediction performance on the CIFAR test dataset while L^1 is about 3% lower. In Figure 5.16 we show the corresponding prediction accuracy on training and test dataset.

5.5.5 Complexity Measure is Data Insensitive

A model complexity measure should be insensitive to data. That is, in the case of sampling two different datasets sampled from the same data distribution under i.i.d, the model complexity measured by LANN on these two datasets is expected to be similar.

To verify if our complexity measure by LANN is data sensitive, we measure the approximation error of LANNs on the test dataset. Below in Table 5.3 we compare approximation errors on the training dataset (the dataset used to build LANNs) and test dataset of models

Dataset	Model	$\mathcal{E}_{train}$	$\mathcal{E}_{test}$
MNIST	L3M100$_T$	0.0999	0.0988
MNIST	L6M100$_T$	0.0979	0.0971
MNIST	L3M200$_T$	0.0911	0.0907
MNIST	L3M300$_S$	0.0944	0.0942
MNIST	L3M300$_T$	0.0989	0.0977
CIFAR	L3M200$_T$	0.0979	0.0984
CIFAR	L6M200$_T$	0.0973	0.0970
CIFAR	L3M400$_T$	0.0984	0.0976
CIFAR	L3M(768,256,128)$_T$	0.0970	0.0979

Table 5.3: Compare approximation error on training dataset and test dataset.

used in our experiments. The results show that LANNs achieve very close approximation error on training and test dataset, which demonstrates that our complexity measure is data dependence but data insensitive.

Furthermore, our experiment shows that the number of linear function pieces generated on each layer of a LANN g is always almost uniform. That is, there is no obvious difference between the number of subfunction pieces, written as k_i, of early hidden layers and later hidden layers. This result is consistent with our discussion of Eq. (5.20). For a neuron from an early hidden layer, the amplification coefficient $w_{i,j}^{(e)}$ is exponentially larger than that of a neuron from a later layer. On the other hand, for a neuron for a later layer, the unignorable error $\mathbb{E}[\hat{\epsilon}_{i,j}]$ is exponentially larger than that of a neuron from an early layer due to layerwise accumulation. The approximations of neurons from early and later layers are equally important and small values of $\mathbb{E}[e_{i,j}]$ are desired for both. In this case, we expect the number of linear function pieces in each layer to be uniformly distributed.

5.6 Interpretation by LANN

In this section, we demonstrate that LANN is able to provide interpretations to the predictions of the target model. A LANN establishes a piecewise linear approximation of the target model with guaranteed approximation degree, then the *OpenBox* method can be used to directly provide interpretations based on the piecewise linear approximation.

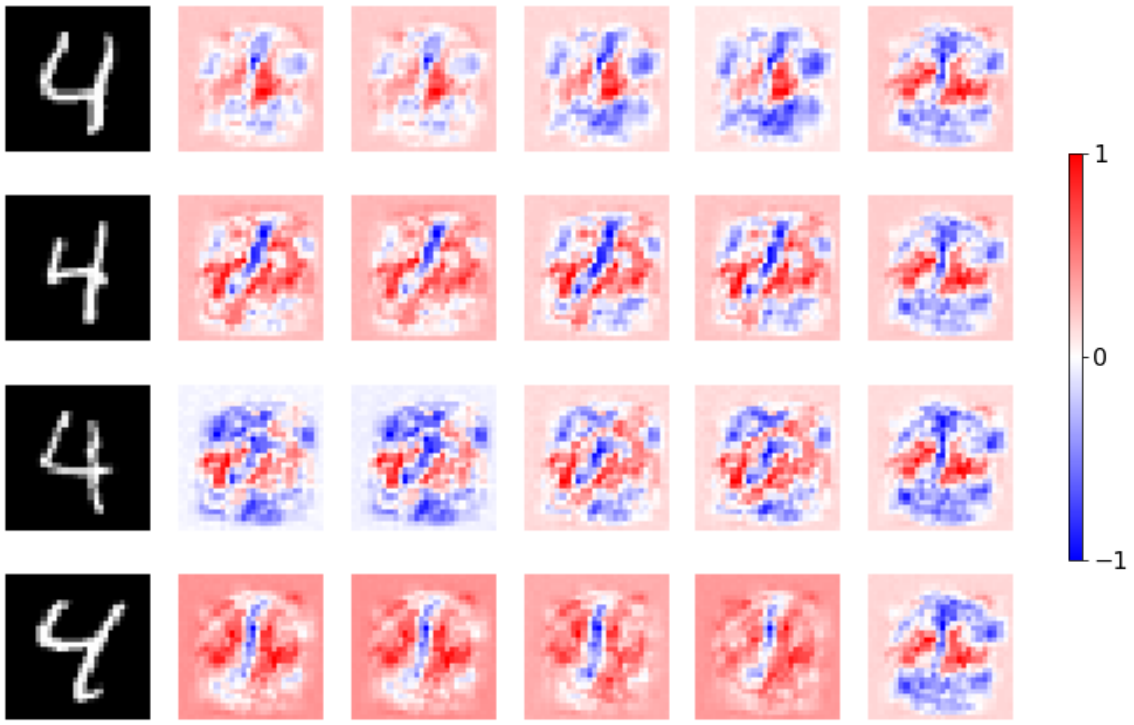


Figure 5.17: Interpretations of the example images from MNIST with the class label '4'. The first column is the original input image. The second column shows the interpretations given by Vanilla Gradient [114]. The third to sixth columns are the interpretations provided by LANN built with different values of λ: 0.02, 0.05, 0.1, 0.2.

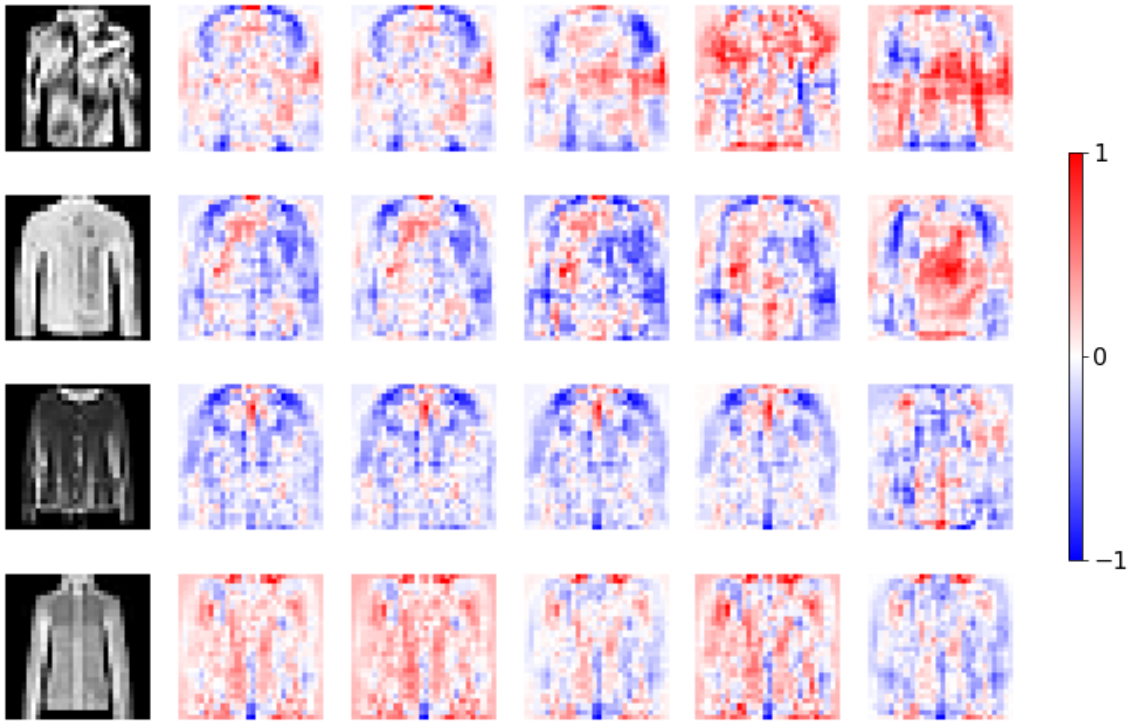


Figure 5.18: Interpretations of the example images from FMNIST with class label 'coat'. The first column shows the original input images. The second column shows the interpretations given by the Vanilla Gradient. The third to sixth columns are the interpretations provided by LANN built with different values of λ: 0.02, 0.05, 0.1, 0.2.

Suppose a LANN g is built from the target model f under certain approximation error λ. Given an input x, the linear region including x is determined by the activation pattern. Its linear classification function $y = Wx + b$ is computable using the activation pattern (Equation 5.3). Things go easily under the linear situation. The magnitude of the weight matrix W describes the importance of every feature to every class label. Specifically, the i-th row, denoted by W_i, indicates the importance of features to the class label i.

Given an input x, a LANN obtains a linear approximation of the curvilinear decision boundary corresponding to the linear region including x. The idea of constructing a linear approximator at the neighborhood of an input sample then interpreting using the linear approximator is a common idea to provide interpretation to curve neural networks [107, 114, 115]. One approach of constructing such a linear approximator is by first-order Taylor expansion [114]. Another kind of approaches are perturbation-based, that is to perturb a group of points around the input sample x, then learn a local linear approximator [107] or

compute the average value of their first-order Taylor derivative [115]. The former approach can be considered a linear approximator within a very tiny Δx ($\lim \Delta x \to 0$) distance around x. It is efficiency but provide interpretations which are sensitive to perturbations [42]. The second approach provides smoother and more stable interpretations but it spends more time to give an interpretation.

Given an input x, LANN generates interpretation in one forward propagation time cost, which is to compute the corresponding linear function (see OpenBox). As well, interpretations provided by a piecewise linear approximation (i.e., LANN) stays consistent within the linear region including x. With different λ, the average volume of linear regions changes. In this case, the value of λ affects the volume of linear regions within which the interpretations stay stable. Another effectiveness of the stable linear region is, the linear approximator reflects the common characteristics of the region and ignores the unique characteristics of every point. This acts as a smoothness within the linear region, and provides a regional smooth interpretation. In this case, with the agreement of large λ, the average volume of linear regions grows, and the interpretation provided by a LANN is able to reflect the importance of features in a large region.

We show that a LANN network can efficiently provide smooth interpretations.

In Figure 5.17 and 5.18 we visualize interpretations given by a LANN, comparing with interpretations given by Vanilla Gradient, on several arbitrarily selected images. To investigate the effects of the value of λ on interpretations, LANN networks are built with different values of λ. The results show, with a small value of λ, interpretations given by LANN look close to that given by Vanilla Gradient (the third column). That agrees with our discussion above that the extremity of a linear region reducing is a point, thus with a small value of λ, the interpretation to the linear region tends to the interpretation of the point. See the figures from left-to-right order, with a larger value of λ, the interpretations become smoother with fewer noises and tend to align with image shape. This shows that, for a linear region with a larger volume, it is more possible to capture those commonly important features.

5.7 Summary

We studied the complexity measure of deep fully-connected neural networks with smooth curve activation functions. We first proposed the linear approximation neural network (LANN), which can approximate the function of a given DNN model to a required approximation degree with as a small number of linear regions as possible. We discussed the number of linear regions formed by a LANN, and use it as a measure of model complexity. From the view of complexity measure, we estimate the amplification of small perturbations

on a neuron to impact the output and analyzed the occurrence of overfitting. Our results demonstrated the positive correlation between the growth of model complexity and occurrence of overfitting in the training process, and L^1, L^2 regularizations which are designed for preventing overfitting showed their impacts on constraining model complexity. Based on this discovery, we developed two approaches for preventing overfitting: neuron pruning and customized L^1 regularization.

Our proposed complexity measure applies to any deep fully connected neural networks with smooth curve activation functions. For example, our approach is applicable to fully connected neural network with Batch Norm [64] layers. The contribution of a Batch Norm layer $\gamma\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta$ to our approach can be considered as a coefficient γ/σ to the error propogation equation.

There are some future works to follow up this project. One interesting future direction is to generalize the usage of our proposed linear approximation neural network (LANN) to other types of network structures (i.e., CNN, RNN). Another interesting future direction is to investigate the complexity of data samples applying the complexity of models measured by our approach. The intuition is, with higher sample complexity, the trained model might be with higher effective model complexity. In this case, the effective model complexity can reflect the complexity of the data on which it is trained.

Chapter 6

Conclusion and Future Directions

Due to the rapid development of deep neural networks in recent years, it is essential for us to better understand deep learning models. The piecewise linear property provides a way to explore to understand deep neural networks. This thesis focuses on two typical deep learning understanding problems, namely model interpretability and model complexity, and proposes solutions to these two problems from the perspective of the piecewise linear property.

In this chapter, we summarize the research presented in this thesis, then discuss several future directions in understanding deep learning models from the perspective of piecewise linear property.

6.1 Summary of the Thesis

In this thesis, we study the understanding of deep neural networks from the perspective of piecewise linear property. We first provide a comprehensive overview of the piecewise linear property, which is achieved by analyzing the network structures of deep neural network models with piecewise linear activation functions (e.g., ReLU, hard Tanh). We discuss the advantages and important roles played by the piecewise linear property in understanding deep neural networks. Then, we investigate the understanding problems of model interpretability and model complexity by adopting the piecewise linear property. We develop two model interpretation approaches to provide local interpretations for predictions made by deep neural networks. We also develop an approach to measure the model complexity of deep neural networks. The summary of this thesis is as follow.

We investigate the model interpretation from the perspective of piecewise linear property:

- We propose an exact and consistent interpretation to deep neural network with piecewise linear activation function, by analyzing the piecewise linear property of such

model architecures. By studying the states of hidden neurons and the linear regions generated by a piecewise linear neural network, we prove that a piecewise linear neural network is mathematically equivalent to a set of local linear classifiers, which can be efficiently computed by the proposed OpenBox method. Our experimental results show that the decision features and the polytope boundary features of local linear classifiers provide exact and consistent interpretations on the overall behavior of a piecewise linear neural network. Such interpretations are highly effective in hacking and debugging piecewise linear neural network models.

- We propose a mimic learning approach to provide interpretation to arbitrary deep learning models. We design the oblique model tree as a mimic model, which is a tree-structured classification model. The oblique model tree acts by dividing the input space into a number of disjoint regions and proposing a local logistic regression model within each region. We propose an algorithm to efficiently build an oblique model tree from the given deep neural network. The iterative space divisions and continuous optimization of local logistic regressions provide enough capacity for the oblique model tree to mimic complex deep neural networks. We show that the interpretation to predictions of the deep model can be given by analyzing the tree structure and local linear regression models of a mimic oblique model tree.

In general, our proposed OpenBox and OMT fit different application scenarios. Specifically, if given a piecewise linear neural network whose hidden states and parameter values are reachable, OpenBox is a good choice to provide trustworthy interpretations to predictions efficiently. If given a non-piecewise linear neural network or the hidden states are unreachable, OMT can be used to provide interpretation by approximation. However, the interpretation given by OpenBox is intuitively more trustworthy and precise than interpretations given by OMT, since OpenBox provides interpretations based on the closed-form expression of piecewise linear neural networks, while OpenBox provides approximation-based interpretations.

We also investigate model complexity from the perspective of piecewise linear property: In recent years, the piecewise linear property has been adopted to investigate model complexity [90, 104], generalization [97] and robustness [25]. Most of these studies are based on deep neural networks with piecewise linear activation functions. However, there is still limited research on understanding deep neural networks with smooth curve activation functions. Motivated by this, we investigate the model complexity of deep neural networks with smooth curve activation functions.

- We propose a model complexity measure approach to deep neural networks with smooth curve activation functions. We design a framework to make neural networks with curve activation functions benefit from the piecewise linear property. We propose a piecewise linear approximation of deep neural networks with curve activation functions, namely the piecewise linear neural network. We analyze the approximation degree of the piecewise linear approximation model to a network with curve activation functions, in view of the approximation error in layer propagation. Then, we provide an efficient algorithm to build the piecewise linear approximation. The piecewise linear approximation gives deep neural networks with curve activation functions the piecewise linear property. We also measure the model complexity and provide an interpretation analysis to networks based on the piecewise linear approximation.

6.2 Future Directions

Piecewise linear property is a valuable perspective to explore the understanding of deep learning models. In addition to the exploration of model interpretability and model complexity, which we discussed in this thesis, there is some other worth exploring problems and applications. For example, can we obtain an average interpretation of a given specific group of data with some common features? Such a group of data may correspond to a set of neighbor linear regions in the input space. Also, although the exact number of linear regions generated by a piecewise linear neural network is unable to be calculated, can we come out an estimation of the average number of linear regions, as a representation of the effective complexity measure? We will continue to explore these important future directions. Below we introduce two of these future potential directions in detail.

6.2.1 Constraining Model Complexity of Deep Learning Models during Training.

In Chapter 2, we discuss the study of model complexity of deep neural networks and then propose a model complexity measure for deep neural networks with curve activation functions in Chapter 5. In these discussions, we demonstrate the necessity of studying model complexity in deep learning. However, understanding deep learning model complexity is still at an early stage, and more exploration is expected.

Constraining model complexity during the training process is a meaningful and interesting future direction. Deep learning models are always over-parameterized and have a high expressive capacity. A deep learning model trained on a given task may have a high effective complexity, even to the point of redundancy. Preventing the model from being unnecessary high complexity is expected to bring many benefits, including increasing the interpretabil-

ity of the model, preventing the model from the occurrence of overfitting, and making the model easier to compress and migrate, among others. Thus constraining model complexity may be helpful for understanding deep neural network.

However, one important challenge is that constraining model complexity during training calls for a feasible representation of effective model complexity. This requires the complexity measure metric to be able to distinguish situations in which the model is trained twice to zero training error but has a different effective model complexity. In addition, the time cost to measure model complexity should be low and avoid adding a high time cost to the training process.

We look forward to finding a better model complexity constraint approach than existing regularization methods which implicitly constrain model complexity. Such an approach is expected to not increase the difficulty of training the model. High-efficiency model training and model complexity constraints must be able to proceed at the same time. On the other hand, the degree of complexity constraint should be more than existing regularization approaches.

Here we propose a possible idea to implement model complexity constraints during the training of models with piecewise linear activation functions, which is based on the piecewise linear property. Taking the number of linear regions as a representation of model complexity, constraining model complexity means constraining the number of linear regions generated in the input space. Thus, the challenge is in representing the number of linear regions during training and deciding how to constrain them during training.

6.2.2 Measuring Model Complexity by the Volume of Information

In this thesis, we demonstrate the necessity of exploring model complexity in deep learning. We note that the study of the model complexity of deep learning models is still at an early stage, and that there have been limited explorations until now. Proposing a feasible model complexity measure is a promising direction. In Chapter 2 and our survey paper, we group deep learning model complexity into two major problems: the expressive capacity and the effective complexity. From these two problems, we are most concerned with proposing a feasible measure for effective complexity.

We are working on potential ways to measure effective model complexity. One idea is that effective complexity can be represented by the volume of information that a model contains. The minimum description length principle [108] states that "the best model on a given dataset is the one that minimizes the combined cost of describing the model and describing the mismatches between the model and the data." Early in the 1990s, Hinton and Camp [59] applied the minimum description length principle to neural networks. The authors

represented the amount of information in a neural network using the minimum description length of the model. They proposed keeping a neural network simple by constraining the amount of information the network contains.

Since the model structure of neural networks has greatly changed in recent decades, their methods [59] cannot be directly applied to exploring the amount of information in today's significantly more complex deep neural networks. However, we suggest that finding a method to represent the amount of information (i.e. a minimum description length) in deep neural networks would be a good avenue for studying effective model complexity.

Bibliography

[1] Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. Analyzing the behavior of visual question answering models. *arXiv:1606.07356*, 2016.

[2] Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural computation*, 8(3):643–674, 1996.

[3] William W Armstrong, Andrew Dwelly, Jian Dong Liang, Dekang Lin, and Scott Reynolds. Learning and generalization in adaptive logic networks. In *Artificial neural networks*, pages 1173–1176. Elsevier, 1991.

[4] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

[5] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[6] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[7] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting blackbox models via model extraction. *arXiv:1705.08504*, 2017.

[8] Yoshua Bengio and Olivier Delalleau. On the expressive power of deep architectures. In *International Conference on Algorithmic Learning Theory*, pages 18–36. Springer, 2011.

[9] Monica Bianchini and Franco Scarselli. On the complexity of shallow and deep neural network classifiers. In *ESANN*, 2014.

[10] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[11] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[12] Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[13] Nicola Bulso, Matteo Marsili, and Yasser Roudi. On the complexity of logistic regression models. *Neural computation*, 31(8):1592–1623, 2019.

[14] Chunshui Cao, Xianming Liu, Yi Yang, Yinan Yu, Jiang Wang, Zilei Wang, Yongzhen Huang, Liang Wang, Chang Huang, Wei Xu, et al. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2956–2964, 2015.

[15] RJ Caron, JF McDonald, and CM Ponic. A degenerate extreme point strategy for the classification of linear constraints as redundant or necessary. *JOTA*, 62(2):225–237, 1989.

[16] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

[17] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, et al. Interpretability of deep learning models: a survey of results. DAIS, 2017.

[18] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. Distilling knowledge from deep networks with applications to healthcare domain. *arXiv preprint arXiv:1512.03542*, 2015.

[19] Zhengping Che, Sanjay Purushotham, Robinder G. Khemani, and Yan Liu. Interpretable deep models for icu outcome prediction. In *AMIA*. AMIA, 2016.

[20] Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Ekaterina Gonina, et al. State-of-the-art speech recognition with sequence-to-sequence models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4774–4778. IEEE, 2018.

[21] Jan Chorowski and Jacek M Zurada. Learning understandable neural networks with nonnegative weight constraints. *TNNLS*, 26(1):62–69, 2015.

[22] Lingyang Chu, Xia Hu, Juhua Hu, Lanjun Wang, and Jian Pei. Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *Pro-*

ceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1244–1253. ACM, 2018.

[23] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

[24] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Shilad Sen, Werner Geyer, Jill Freyne, and Pablo Castells, editors, *RecSys*, pages 191–198. ACM, 2016.

[25] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. In *the 22nd International Conference on Artificial Intelligence and Statistics*, pages 2057–2066. PMLR, 2019.

[26] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[27] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

[28] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[29] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[30] Patrick Doetsch, Christian Buck, Pavlo Golik, Niklas Hoppe, Michael Kramp, Johannes Laudenberg, Christian Oberdörfer, Pascal Steingrube, Jens Forster, and Arne Mauser. Logistic model trees with auc split criterion for the kdd cup 2009 small challenge. In Gideon Dror, Mar Boullé, Isabelle Guyon, Vincent Lemaire, and David Vogel, editors, *Proceedings of KDD-Cup 2009 Competition*, volume 7 of *Proceedings of Machine Learning Research*, pages 77–88, New York, New York, USA, 28 Jun 2009. PMLR.

[31] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *CVPR*, pages 4829–4837, 2016.

[32] Simon S Du and Jason D Lee. On the power of over-parametrization in neural networks with quadratic activation. *arXiv preprint arXiv:1803.01206*, 2018.

[33] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940, 2016.

[34] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[35] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Stefano Soatto. Classification regions of deep neural networks. *CoRR*, abs/1705.09552, 2017.

[36] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *arXiv:1704.03296*, 2017.

[37] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[38] Brendan J Frey and Geoffrey E Hinton. Variational learning in nonlinear gaussian belief networks. *Neural Computation*, 11(1):193–213, 1999.

[39] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Annals of statistics*, 28(2):337–407, 2000.

[40] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In Tarek R. Besold and Oliver Kutz, editors, *CEx@AI*IA*, volume 2071 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[41] Hongyang Gao and Shuiwang Ji. Graph representation learning via hard and channel-wise attention networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 741–749, 2019.

[42] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3681–3688, 2019.

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

[44] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *ICAIS*, pages 315–323, 2011.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[46] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

[47] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57, 2017.

[48] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[49] Ingo Gühring, Gitta Kutyniok, and Philipp Petersen. Complexity bounds for approximations with deep relu neural networks in sobolev norms. 2019.

[50] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[51] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. *arXiv preprint arXiv:1901.09021*, 2019.

[52] Nick Harvey, Chris Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension bounds for piecewise linear neural networks. *arXiv:1703.02930*, 2017.

[53] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[54] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the selection of initialization and activation function for deep neural networks. *arXiv preprint arXiv:1805.08266*, 2018.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[57] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. *CoRR*, abs/1603.08507, 2016.

[58] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. *Coursera, video lectures*, 264:1, 2012.

[59] Geoffrey Hinton and Drew van Camp. Keeping neural networks simple by minimising the description length of weights. In *Proceedings of COLT-93*, pages 5–13.

[60] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

[61] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[62] Patrik O Hoyer. Non-negative sparse coding. In *WNNSP*, pages 557–565, 2002.

[63] Xia Hu, Weiqing Liu, Jiang Bian, and Jian Pei. measuring complexity of deep neural network with curve activation functions. *arXiv*, 2020.

[64] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[65] Paul C Kainen, Vera Kurková, and Andrew Vogt. An integral formula for heaviside neural networks. *Neural Network World*, 10:313–319, 2000.

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

[67] Joe Kilian and Hava T Siegelmann. On the power of sigmoid neural networks. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 137–143, 1993.

[68] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. *arXiv:1711.00867*, 2017.

[69] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv:1703.04730*, 2017.

[70] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[71] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[72] Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Phrase-based & neural unsupervised machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5039–5049, 2018.

[73] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Mach. Learn.*, 59(1-2):161–205, May 2005.

[74] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[75] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[76] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[77] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *NIPS*, pages 801–808, 2007.

[78] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv:1506.01066*, 2015.

[79] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, 40(3):203–228, 2000.

[80] Zachary Chase Lipton. The mythos of model interpretability. *CoRR*, abs/1606.03490, 2016.

[81] Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 414–429. Springer, 2018.

[82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Information Theory*, 28(2):129–136, 1982.

[83] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems*, pages 6231–6239, 2017.

[84] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[85] Wolfgang Maass, Georg Schnitger, and Eduardo D Sontag. A comparison of the computational power of sigmoid and boolean threshold circuits. In *Theoretical Advances in Neural Computation and Learning*, pages 127–151. Springer, 1994.

[86] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, pages 5188–5196, 2015.

[87] Naresh Manwani and P. S. Sastry. Geometric decision tree. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 42(1):181–192, 2012.

[88] Bertrand Michel and Anthony Nouy. Learning with tree tensor networks: complexity estimates and model selection. *arXiv preprint arXiv:2007.01165*, 2020.

[89] Sina Mohseni, Niloofar Zarei, and Eric D. Ragan. A multidisciplinary survey and framework for design and evaluation of explainable ai systems, 2018.

[90] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 2017.

[91] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *NIPS*, pages 2924–2932, 2014.

[92] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[93] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *CoRR*, abs/cs/9408103, 1994.

[94] In Jae Myung. The importance of complexity in model selection. *Journal of mathematical psychology*, 44(1):190–204, 2000.

[95] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[96] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

[97] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations*, 2018.

[98] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[99] Rasmus Berg Palm. Prediction as a candidate for learning deep hierarchical models of data. *Technical University of Denmark*, 5, 2012.

[100] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv:1312.6098*, 2013.

[101] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. Lower bounds on the robustness to adversarial perturbations. In *Advances in Neural Information Processing Systems*, pages 804–813, 2017.

[102] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368, 2016.

[103] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[104] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2847–2854. JMLR, 2017.

[105] Nadeem N Rather, Chintan O Patel, and Sharib A Khan. Using deep learning towards biomedical knowledge discovery. *IJMSC*, 3(2):1, 2017.

[106] Patrick Rebentrost, Brajesh Gupt, and Thomas R Bromley. Quantum computational finance: Monte carlo pricing of financial derivatives. *Physical Review A*, 98(2):022321, 2018.

[107] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[108] Jorma Rissanen. Stochastic complexity and modeling. *The annals of statistics*, pages 1080–1100, 1986.

[109] Nina Schaaf, Marco Huber, and Johannes Maucher. Enhancing decision tree based interpretation of deep neural networks through l1-orthogonal regularization. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 42–49. IEEE, 2019.

[110] Ramprasaath R Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv preprint arXiv:1611.07450*, 2016.

[111] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. pages 3145–3153, 2017.

[112] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–, January 2016.

[113] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[114] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[115] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *Workshop on Visualization for Deep Learning, ICML*, 2017.

[116] David J Spiegelhalter, Nicola G Best, Bradley P Carlin, and Angelika Van Der Linde. Bayesian measures of model complexity and fit. *Journal of the royal statistical society: Series b (statistical methodology)*, 64(4):583–639, 2002.

[117] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv:1703.01365*, 2017.

[118] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[119] Andrea Vedaldi and Karel Lenc. Matconvnet - convolutional neural networks for matlab. *CoRR*, abs/1412.4564, 2014.

[120] Wei-Hung Weng, Yu-An Chung, and Peter Szolovits. Unsupervised clinical language translation. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[121] Mike Wu, Michael C. Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. *AAAI*, 2018.

[122] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[123] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[124] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[125] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *ICML*, pages 325–333, 2013.

[126] Quanshi Zhang, Yu Yang, Ying Nian Wu, and Song-Chun Zhu. Interpreting cnns via decision trees. *CoRR*, abs/1802.00121, 2018.

[127] Quanshi Zhang and Songchun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

[128] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *arXiv:1711.05611*, 2017.

[129] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *CVPR*, pages 2921–2929, 2016.

[130] Jie Zhu, Ying Shan, JC Mao, Dong Yu, Holakou Rahmanian, and Yi Zhang. Deep embedding forest: Forest-based serving with deep embedding features. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1703–1711, 2017.