Interpretability of Feed Forward Neural Network from Activation Perspective

Ziping Jiang, BSc

School of Computing and Communications

Lancaster University

A thesis submitted for the degree of

Doctor of Philosophy

September, 2024

Interpretability of Feed Forward Neural Network from Activation Perspective

Ziping Jiang, BSc.

School of Computing and Communications, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy*. September, 2024.

Abstract

In the past decade, deep neural networks have presented promising results in various fields. To advance their success and to mitigate the limitation of opaqueness, this dissertation explores the explainability of feedforward neural networks from the activation function perspective.

The theoretical outcome of this work is a framework for analyzing the neural network that was developed on the basis of existing literatures. This framework generalizes the definition of activation pattern by indexing neurons with an index family and releasing the constraint on activation functions. Based on this framework, this research identifies a novel *dying neuron issue* that prevents networks from reaching their optimum by studying the learning dynamic of neural networks. To further understand the *dying neuron* issue, two metrics are proposed to explore expressive ability of models. The *pattern similarity* records the overall severity of *dying neuron issues* of neural network for comparison across models, while the *neuron entropy* measures the volatility of single neurons for understanding the unit-wise model behaviour in a model. Apart from the expressive ability, this work also investigates the robustness of models by decomposing the computational graph of neural network using the proposed framework. In particular, it shows that the unsecured data can be categorized into *Lipschitz vulnerability* and *float vulnerability* according to the source of instability.

Based on the insights of theoretical analysis, this work introduces two downstream applications of the proposed framework. The *neuron entropy pruning* (NEP) computes the importance score of parameter by integrating the neuron entropy and

removes the unimportant parameters to reduce the model scale. The smoothed classifier with reformed float path in dual direction (SCRFP-2) reforms the training and prediction based on the smoothed classifier such that it increases the robustness of the model. Both of the models outperform the benchmark, which further supports the theoretical analysis presented by this work.

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis does not exceed the maximum permitted word length of 80,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: 48889

Ziping Jiang

Publications

Contribution publications

Z. Jiang, "On explaining neural network robustness with activation path," in *The Eleventh International Conference on Learning Representations*, 2023

Z. Jiang, Y. Wang, C.-T. Li, P. Angelov, and R. Jiang, "Delve into activations: Towards understanding dying neuron," *IEEE Transactions on Artificial Intelligence*, 2022

Contribution Statement

This dissertation is developed during my Ph.D. study. As the project focuses on explainable AI, the research undertaken is consistent and have several outcomes published during the past several years. As the first author of all the related publications, the analysis and discussion presented in this dissertation is my own work.

The most recent work is a submission to IEEE transaction on Pattern Analysis and Machine Intelligence (TPAMI)'2023 under single-blind review. Parts of theoretical analysis is displayed in Chapter 4, Chapter 5 and Chapter 6.

The most recent publication is an International Conference on Learning Representations (ICLR)'2023 paper [1] that discusses the robustness of neural network, which is partly presented in Chapter 4 and Chapter 6.

Prior to this work, a publication at IEEE Transactions on Artificial Intelligence journal [2] investigates the expressive ability of neural network. The results are partly presented in Chapter 3.

There are several publications of my own work that is out of the scope of explainable AI, but relevant to the deep learning models, which are published at Recent advances in AI-enabled Automated Medical Diagnosis [3] and IEEE Access journal [4]. Parts of the related work sections are presented in Chapter 2.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation aims to provide insights into neural network explainability from the perspective of activation functions. To better understand how activation functions affect the performance of neural networks, this work introduces a framework for analyzing neuron-level behavior and connects it with model performance. Following the research trajectory of previous works, this study begins by addressing the observation that the performance of deep neural networks with different activation functions varies, even under the same structure and initialization. With the proposed framework, it further presents a theoretical investigation of the expressive ability and robustness of neural networks. Moreover, it explores downstream applications built on the insights provided by the proposed investigation.

As an introduction, the rest of this chapter will cover: (1) the importance of neural network explainability and the motivation for this research, (2) the research questions and contributions of this work, and (3) the structure of this dissertation.

1.1 Motivation and Research Questions

The research presented in this dissertation focuses on the explainability of deep neural network. Before delve into details, it is necessary to understand (1) what is explainability, (2) why explainability is important in this field and (3) what problem

this dissertation aims to address, which are presented in the following.

1.1.1 Explainability and Interpretability

The research into the explainability and interpretability of machine learning models has emerged alongside the increasing complexity of these models. However, the terms explainability and interpretability, and sometimes other synonyms such as understandability and intelligibility, are often used interchangeably. In general, all these efforts aim to better understand how machine learning models work and improve existing models. Therefore, it is challenging to separate these terms with mathematically rigorous definitions.

One widely accepted definition of interpretability is considered to be the ability to provide an explanation or to present in understandable terms to a human [5], [6]. In other words, interpretability entails the understanding of internal functions and characteristics with meaningful, clear, and logical reasoning. Prior to deep learning models, classic statistical models, such as decision trees, hidden Markov models, and Gaussian mixture models, are deemed to be interpretable as one can easily understand how the outputs of those models are computed by following explicit logical rules designed for those models.

On the contrary, most existing works categorize explainability as post hoc analysis performed on the features and representations of existing models to gain insights from the model [7], [8]. Research into explainability focuses on describing and investigating the processing and representation of data inside the black box. In recent years, state-of-the-art deep neural networks have millions or even billions of parameters. It is no longer trivial to justify why a high-resolution picture of a dolphin is correctly recognized among 1,000 classes by a 19-layer model. During the progress of deep learning, researchers have provided post hoc analysis of model features, such as the distribution of intermediate layer outputs and learning speeds, to illustrate how different aspects of the model affect its performance, thereby refining existing models.

Adhering to the terms described above, this work explores the explainability of neural networks. With the objective of understanding how the activation function affects feedforward neural networks, this work evaluates the performance of neural networks using proposed metrics that summarize the features of intermediate results of the model in a post hoc manner.

1.1.2 The Importance of Explainability

Promoted by the progress of hardware facilities, research in machine learning has made astonishing progress in various fields, from computer vision to natural language processing [9]–[11]. On the other hand, as model performance improves, model scale also grows rapidly. This means that it is even more challenging for researchers to explain how the individual neurons work together to approximate an objective function and arrive at the final output. In other words, machine learning models are now often viewed as black boxes. This opaqueness of machine learning models makes it difficult to apply them to specific tasks with high misprediction costs, such as medical diagnosis, economic forecasting, and automated vehicles. Lacking awareness of what is happening inside the box gradually becomes a barrier to utilizing its potential.

Besides being desirable on its own account, explainability also often helps researchers understand the bottleneck and further improve the performance of deep learning models. One of the most famous examples is the discussion of the vanishing and exploding gradient issue. In the early days of neural networks, the Sigmoid function was widely used as a non-linear activation function. However, such a structure fails to converge as the models become deeper. The gradient vanishing issue of the Sigmoid activation refers to the phenomenon that when the inputs of neurons are extremely large or small, the Sigmoid function saturates at regions where gradients are almost zero. This results in a failure of the back-propagation of the network. Since it was proposed, it was widely accepted as the cause of the poor performance of Sigmoid and other activations, such as the tanh function. This

discussion has led to the proposal of the rectified linear unit (ReLU) activation, which is one of the cornerstones of deep learning, and dramatically boosts the development of machine learning.

Another critical aspect of explainability is that it provides solutions to downstream applications and continuously improves machine learning models. For instance, following the discussion of the vanishing/exploding gradient problem, it was observed that the accuracy of a convolutional neural network would decrease rapidly when the depth of the network increases, despite the use of input regularization techniques. This problem was referred to as the degradation issue [12], [13]. The investigation of the degradation issue suggests that the training of a deep network not only depends on its weights but also on its structure. Inspired by the results, the residual learning framework was introduced to minimize the network error using residual mapping instead of the original mapping [14] during training.

1.1.3 Motivation

One of the most prominent arguments in the deep learning field is the *universal approximation theorem*, which posits that any arbitrary objective function can be approximated by a neural network with a certain depth or width to any desired precision [15], [16]. However, empirical studies demonstrate that the choice of activation can significantly impact the potential performance of a model.

The most notable case of model performance gaps caused by the activation function can be observed from the comparison between ReLU networks and Sigmoid networks. [17]. This disparity is widely attributed to the *vanishing gradient issue*, stemming from the over-saturation of hidden units. Since the discovery of this issue, many techniques, such as normalization layers and weight initialization, have been proposed from various perspectives with the aim of addressing it.

Normalization layers, such as Batch Normalization, standardize inputs of each layer to have zero mean and unit variance, mitigating issues like vanishing or exploding gradients by preventing activations from becoming too extreme. This

continuous adjustment during training helps maintain stable activation distributions and speeds up convergence by reducing internal covariate shift.

Weight initialization, in contrast, sets the initial conditions for the weights of neural network to optimize the propagation of gradients during training. Strategies like He and Xavier initialization consider the activation functions and the architecture to adjust the variance of initial weights, ensuring that gradients neither vanish nor explode as they propagate through deeper layers [17], [18]. While weight initialization provides a conducive starting point for training, normalization layers dynamically adjust the data flow between layers to ensure ongoing stability and efficiency in learning. Together, these methods enhance the robustness and performance of deep learning models.

Despite the use of the above methods, there still exist distinguishable differences between the performance of Sigmoid networks and ReLU networks. This means that there are still undiscovered issues caused by the activation function that can affect model performance. In fact, as most of the components in a feedforward neural network provide only linear mapping from input to output, the ability of a neural network to approximate functions with high complexity is mostly granted by the non-linearity from the activation function. Therefore, it is necessary to investigate the explainability of neural networks from an activation perspective.

This work is based on the above observation and aims to propose a generalized framework that describes the neural network from the activation function perspective.

1.2 Research Questions and Objectives

This section illustrates the connections and progress of research undertaken in this dissertation. The research can be split into three parts that delve into the neural network interpretability step by step with activation function as an entry point:

- Chapter 3 serves as an introductory Chapter that verifies that the performance

of neural networks with different activation functions are different. It further investigates the performance gap on several small models to gain insights on activation functions.

- Chapter 4 lays the groundwork with analytical tools for describing the behavior of neurons in a network and illustrates the basic properties of these tools. Armed with these tools, the following two chapters explore the explainability of neural networks from two aspects.

- Chapter 5 re-examines the introductory question, provides insights into the model's ability to approximate complex functions, and introduces a model pruning algorithm. Chapter 6 investigates the explainability of model robustness, discusses the reason why neural networks are prone to attacks by maliciously designed inputs, and proposes a robust training algorithm.

1.2.1 An Introductory Question

The research in this work is motivated by an intuitive and fundamental question regarding activation functions:

Question 1. *Why do neural networks with Sigmoid activation and ReLU activation exhibit significant performance gaps?*

To address this question, Chapter 3 first re-investigates existing literature. One of the most profound arguments suggests that the performance gap is caused by the *vanishing gradient problem*. It posits that when the inputs of the activation function have large magnitudes, the Sigmoid activation saturates in regions where gradients are almost zero, resulting in a failure of back-propagation in the network. In contrast, ReLU activation, as a piece-wise linear function, has constant partial derivative that preventing the gradient with respect to the loss function from vanishing. Once the vanishing gradient issue was identified, several attempts were made to address this problem.

To validate the effects of those techniques, Chapter 3 compares the performance, weight distribution, and gradients of saturated activation functions with non-saturated activation functions. It is found that the weight distribution no longer clusters at the saturated region, and the gradient remains stable during backpropagation after introducing batch normalization and weight initialization to the Sigmoid network. This suggests that the gradient vanishing problem is addressed by the proposed techniques. However, there is still a distinguishable performance gap between neural networks with different activation functions, implying the existence of undiscovered issues.

Chapter 3 then delves into the training dynamics of networks from a theoretical perspective. It is found that the training of networks with non-piecewise linear activation can be affected by the current weight, making it less stable. To illustrate these results, a toy model is constructed to track the change of weights and the ability to approximate an objective function of each neuron during training, revealing a novel *dying neuron* issue for Sigmoid activation. Unlike the gradient vanishing issue, the gradient of *dead neurons* is non-zero while the model still fails to update. Moreover, it differs from the *dying ReLU issue* in that:

- *Dying neurons* are not limited to ReLU activation.
- *Dying neurons* have non-constant outputs, but the outputs are similar for arbitrary inputs, which barely contribute to the model performance.

Broadly, it can be viewed as a generalized *Dying ReLU* issue with similar effects on model performance but in a wider form.

1.2.2 Analytical Tools

The second research question that follows up on the observed *dying neuron* issue in Chapter 3 is proposed:

Question 2. *How to systematically describe the behavior of activation functions to investigate the dying neuron issue?*

To answer this question, this research introduces analytical tools for studying the explainability of neural networks from an activation perspective in Chapter 4. These analytical tools are developed from the concept of *activation patterns* [19], originally conceived as linear regions to explore the expressive capacity of neural networks with piece-wise linear activation functions, but with expanded properties. The proposed definitions and concepts are devised to characterize the state of neurons for arbitrary activation functions, facilitating a comprehensive examination of neural network performance across models within a broader scope beyond individual activation regions. Additionally, to ensure the completeness of this framework, lemmas and theorems are introduced to describe the properties of activation regions and pattern within arbitrary subspaces.

Given a network $\mathcal{N}$ defined on $\mathbb{R}^n$ with a piece-wise linear activation function π, the input space of $\mathcal{N}$ can be partitioned into a number of regions, where the mapping of $\mathcal{N}$ is linear. Each of the regions was then referred to a linear region. Within each of the linear region $\mathcal{R}$, given $\mathbf{x} \in \mathcal{R}$, the activation status of each neuron remains the same. Conversely, by assigning each neuron a pattern, a unique region $\mathcal{R} \subset \mathbb{R}^n$ can be defined such that for every $\mathbf{x} \in \mathcal{R}$, the activation status of each neuron on x satisfies the pattern. A collection of neuron activation status is referred to as an *activation pattern*, and the corresponding region is formally described as an *activation region*. The following figure illustrates how the input space is splitted into different linear regions by a neural network with 2 hidden layers.

The definitions of activation region and activation pattern provide a way to connect the behavior of the activation function with input space. Intuitively, neural networks with more linear regions have higher capacity in approximating complex objective functions. Therefore, the number of linear regions are then adopted as a proxy of the expressive ability of neural networks [19]. However, there are several gaps in existing works. Chapter 4 aims to address them in the following ways.

First, activation pattern describes the neuron-level status within a region for a given activation function, but fails to generalize this concept across different

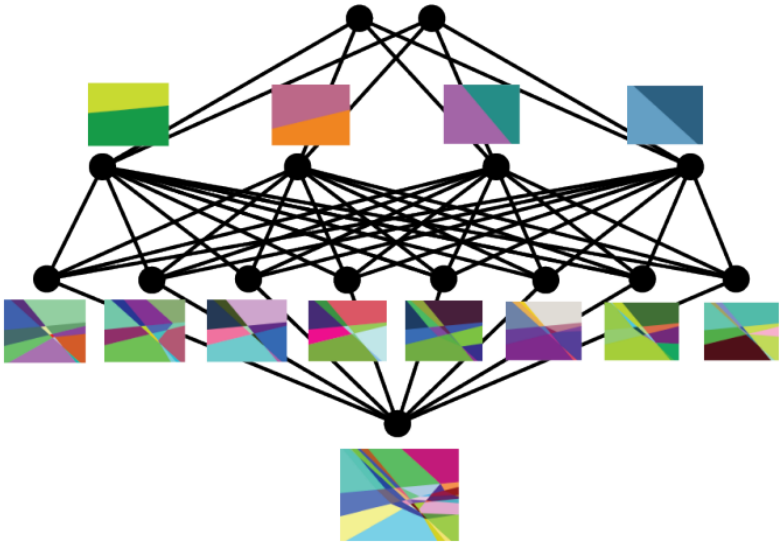


Figure 1.1: An illustration of how input space is splitted into linear regions by a 3 layer neural network with ReLU activation function [19].

activation functions. To answer the introductory question of this dissertation, Chapter 4 extends the definition to arbitrary activation functions and introduces *pattern similarity* to enable comparison between different activation functions.

Second, the number of linear regions reveals the potential of expressive ability of neural networks, while it is not directly linked to the model performance. To mitigate this issue, Chapter 4 further suggests applying a metric named *neuron entropy* to measure the stability of neurons given an activation pattern. It further illustrates their connection with the model performance as well as other metrics proposed in previous works.

At last, it is shown that the average volume of each activation region decreases as the scale of a network increases. This implies that the insights provided by a single region are less informative for deep networks. To analyze the performance of large-scale models, Chapter 4 generalizes the analysis from single linear region to a larger scale by categorizing the neurons into *float/fixed neuron* according to stability of their activation pattern within a subspace $\mathcal{R} \subset \mathbb{R}^n$. To yield a complete

framework, Chapter 4 also provides additional discussion to illustrate several useful properties of the proposed concepts.

1.2.3 Explainability of Neural Networks

The *dying neuron issue* discovered in Chapter 3 is insightful in understanding the introductory question 1. It is presented in the form of pure theoretical analysis of back propagation and illustration from simple models. However, it lacks a connection between the micro-level neural behavior and macro-level model performance. The main theoretical discussion of this dissertation, which are presented in Chapter 5 and Chapter 6 that fill this gap by systematically analyzing how activation functions affect the performance of neural network. In particular, it answers the following questions:

Question 3. *How to evaluate and explain the expressive ability and robustness of neural network with the proposed tools?*

Chapter 5 studies the expressive ability of neural networks from an activation perspective. Expressive ability, in the context of deep learning, is the ability of a neural network to approximate complex functions. Following the common research in this area, this dissertation introduces metrics that are related to model performance to characterize how activation functions affect the expressive ability of neural networks. Theoretical research in Chapter 5 can be divided into two parts that focus on different aspects.

First, it verifies and explains the *dying neuron issue* proposed in Chapter 3 by comparing the pattern similarity of neural networks with different activations. It is found that Sigmoid networks and Tanh networks have significantly higher pattern similarity. This means that for any two data points, the majority of neurons are producing similar post-activation values regardless of the input. Therefore, the practical expressive ability of the network is far below that of the theoretical.

Second, it further investigates the general expressive ability of neural networks

by studying the stability of neuron activation patterns on a global scope. It is shown that that float neurons are able to improve the ability of models to represent complex functions. In fact, if most of the neurons are fixed, the mapping of the network is close to a linear mapping, implying a loss of expressive ability. By measuring the volatility of neurons across layers, it is found that a severe *dying neuron* issue occurs at deeper layers (e.g. layers 11-14 for VGG16 net), where the activation pattern of a majority of neurons remains unchanged regardless of the input.

Chapter 6 explores the robustness of models by illustrating the behaviors of neurons in a local region $B(\mathbf{x}, r)$, where $B(\mathbf{x}, r)$ is a sphere centered at input $\mathbf{x}$ with radius r. We show that a robust model should have fewer float neurons locally to achieve better consistency of the model predictions for an input $\mathbf{x}$ and its neighbor with small perturbation $\mathbf{x} + \epsilon$. It further investigates the vulnerable samples where the prediction between input $\mathbf{x}$ and its neighbor $\mathbf{x} + \epsilon$ differ. By decomposing the computational graph of the network, we show that the vulnerability of the sample results from the high local Lipschitz constant and the variation provided by the float neuron.

1.2.4 Applications

Previous investigations into activation regions have explored a diverse range of topics, including sensitivity and potential network issues, and have laid the theoretical groundwork for this work. However, due to the complexity inherent in neural networks, theoretical insights have remained largely confined to understanding neural network behavior and have not been readily translated into practical applications. Consequently, the subsequent research question naturally arises:

Question 4. *How can the insights gained from theoretical analysis be effectively integrated into practical applications?*

This dissertation addresses this question by demonstrating that theoretical findings from Chapters 5 and 6 can be successfully applied to empirical models.

The first application is a pruning algorithm named *Neuron Entropy Pruning* (NEP), developed from the discussion of model expressive ability. Chapter 5 shows that dead neurons can hardly contribute to the network's prediction. This means that those neurons can be removed with little effect on the performance of the neural network. Based on this idea, NEP records the activation status of neurons during training and prunes the neurons with the lowest importance score. It is shown that by leveraging neuron entropy, the proposed method can efficiently reduce the model size with better accuracy under the same sparsity compared to benchmark models.

The second algorithm introduced in this work is the *Smoothed Classifier with Reformed Float Path in dual direction* (SCRFP-2), which builds on randomized smoothing algorithms and is able to further improve their robustness and accuracy. Chapter 6 decomposes the local mapping function into fixed paths and float paths according to the stability of neurons on the path. The fixed paths have a stable mapping relationship between input and output, while the float paths can result in a sudden change in the mapping function and alter the result. SCRFP-2 stabilizes the float neurons by amplifying their loss during training and repressing it during prediction, therefore achieving better model robustness.

1.3 Structure

There are seven chapters in this dissertation. Chapter 1 is an introduction that presents the motivation, research questions and the necessity of this research. It describes the process of research undertaken in this dissertation by reasoning through the logic and illustrating the connections between all the chapters in this dissertation. Chapter 2 introduces the background of deep learning along with several research threads that are related to this work. The major contributions of this dissertation are presented in Chapter 4 to Chapter 6:

- Chapter 3 raises a question based on empirical experiment results and provides explanations to the observation from several aspects. The experiments of this

chapter show that there exists a *dying neuron issue* for deep neural network.

- Chapter 4 introduces the framework proposed in this dissertation, with a focus of explaining the motivation and illustrating the properties of the concepts.

- Chapter 5 further explores the *dying neuron issue* by measuring the expressive ability of the neural network with two novel metrics. The insights from this chapter lead to a pruning method that removes the unnecessary parameters and reduces the scale of the model with minimal costs on performance.

- Chapter 6 presents the investigation of model robustness with the proposed framework. In particular, it demonstrates how the perturbation affects model prediction and connects the model robustness with neuron-level response of the network. Based on the discussion, a randomized smoothing based algorithm is proposed to further improve the model robustness.

At last, Chapter 7 concludes this dissertation by summarizing the findings, discussing the future works.

Chapter 2

Background and Related Work

This dissertation focuses on the explainability of deep learning models. In particular, it aims to understand the performance gaps caused by the choice of activation functions and bring insights to downstream applications. This chapter presents the literature review around the related topics to align with the objective.

Section 2.1 discusses the history of machine learning. It starts from presenting the classic methods from statistical learning models, and then focuses on the reasoning the deep learning trends in past decades as well as explaining the bottleneck that prevents it from wider deployment. This illustrates the importance of delving into explainability.

After the general introduction of the related research, Section 2.2 discusses several deep learning topics that are relevant to this dissertation. It reviews the model structures of deep learning models, the development of activation functions, and the previous attempts to explain the deep neural network. At last, Section 2.3 investigates downstream applications of neural networks, including robust training, smoothed classifier, neural network pruning techniques and other techniques that are briefly discussed in this dissertation.

2.1 History of Machine Learning

In the past decade, machine learning has been one of the most striking topics in computer science. Applications based on machine learning models have been deployed into various fields, such as object recognition [20], [21], action detection [22], face recognition [23], natural language processing [24], [25] and generative models [26], [27]. This greatly affects the daily life of the majority population regardless of their knowledge of the underlying models. Due to the expendability of neural networks, deep models with millions of parameters has better potential in approximating objective function with high complex, which grants it ability to solve complicated problems [15], [28].

The essence of machine learning models is a well-designed algorithm that abstracts a real-world task by representing the problem with an objective function [29]. The performance of such a model is measured by a loss function that represents the difference between model prediction and the observation [30]. Given enough samples, the algorithm aims to minimize the loss function by updating the model parameters with backward propagation such that they can better describe the relationship between input data and the corresponding outout [31]. In particular, models are proposed to describe the relationship between observations and targets, while a loss function is introduced to quantify the performance of models. By minimizing the loss function, the parameters of models are then optimized to provide the best estimation based on observations.

2.1.1 Statistical Models

Before the age of deep learning, statistical learning models aims to describe the relationship between observations and targets with statistics and functional analysis [32], [33]. This section introduces several popular machine learning models, including naive Bayes classifier, Gaussian mixture model, support vector machine, hidden Markov model and decision tree. At last, it summarizes the pros and cons of those

methods.

2.1.1.1 Naive Bayes Classifier

The Naive Bayes (NB) classifier is a cornerstone of probabilistic modeling in machine learning, well-known for its simplicity and effectiveness, particularly in text classification tasks. The model is based on Bayes' theorem and assumes that all features contribute independently to the probability of a given outcome, which is known as the "naive" assumption. Given a set of features X, the Naive Bayes classifiers use Bayes' theorem to compute the posterior probability of a class C given a set of features X:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}. \tag{2.1}$$

Given the independence assumption, the likelihood term $P(X|C)$ can be decomposed as the product of the conditional probabilities of the individual features:

$$P(X|C) = \prod_{i=1}^{n} P(X_i|C). \tag{2.2}$$

In addition to Naive Bayes classifier, Gaussian Naive Bayes is commonly used for continuous data, assuming that the data within each class is normally distributed [34]. Multinomial Naive Bayes is well-suited for discrete data, such as word counts in text classification [35]. Bernoulli Naive Bayes is ideal for binary/boolean features, often used in document classification where the presence or absence of a word is significant [36].

In terms of applications, Bayes classifiers are widely applied in various domains, particularly in text classification, such as spam detection [37], sentiment analysis [38] and document categorization [39]. The model's ability to handle large feature spaces efficiently is one of its key strengths, especially in text classification tasks, where the dimensionality is typically very high.

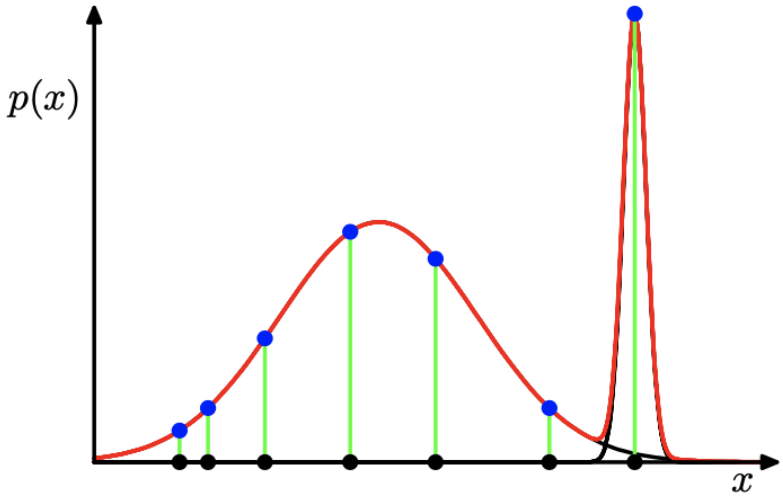


Figure 2.1: An illustration of Gaussian Mixture Model with 2 Gaussian components [40].

2.1.1.2 Gaussian Mixture Models

Gaussian Mixture Models (GMMs) are a powerful probabilistic tool used for modeling the presence of subpopulations within an overall population, where each subpopulation is represented by a Gaussian distribution:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k), \tag{2.3}$$

where K is the number of Gaussian components, π_k is the mixing coefficient, μ_k and Σ_k are the mean vector and covariance matrix of the k-th Gaussian component [29]. The expectation maximization (EM) algorithm is typically used to estimate the parameters of a GMM model by maximizing the likelihood of the observed data [41].

GMMs have found applications across various fields due to their flexibility in modeling data. In clustering, GMMs provide a soft clustering approach, where each data point is assigned a probability of belonging to each cluster, in contrast to hard clustering methods like K-means [42]. This probabilistic approach is

particularly useful in applications like image segmentation, where GMMs can model the distribution of pixel intensities to segment an image into distinct regions [43].

In speech recognition, GMMs are used to model the distribution of acoustic features. Each phoneme or word can be represented as a mixture of Gaussians, capturing the variability in speech signals [44]. GMMs are also employed in anomaly detection by modeling the normal data distribution and identifying data points that do not fit this distribution as anomalies [45].

2.1.1.3 Support Vector Machine

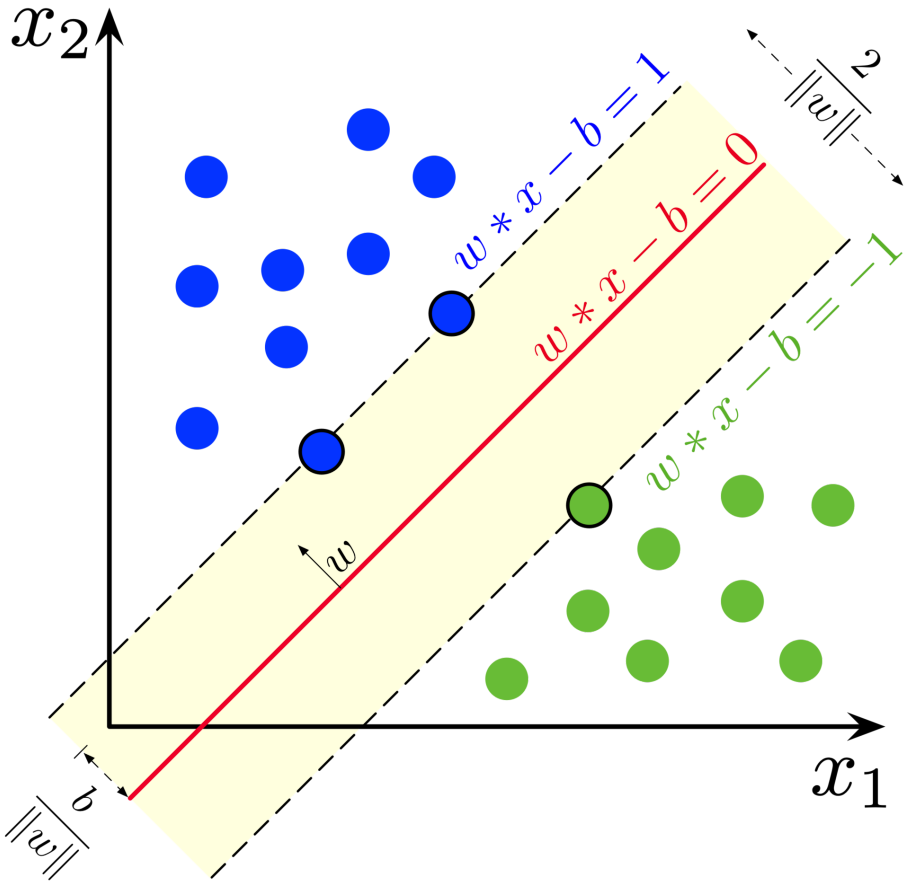


Figure 2.2: An illustration of support vector machine that separate two classes by maximizing the margin with hyperplane [40].

The support vector machine (SVM) is a classification algorithm that designed to

find the optimal hyperplane that separates data points into different classes [46]–[49]. The idea behind SVM is to identify a hyperplane that maximizes the margin between different classes, which is the distance between the hyperplane and the nearest data point from each class. For a binary classification problem, given a set of training examples $\{(x_i, y_i)\}$ where $x_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$, the SVM aims to solve the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2, s.t. y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i, \tag{2.4}$$

where $\mathbf{w}$ is the weight vector and $\mathbf{b}$ is the bias term. Based on this idea, many modifications are then proposed to generalize it from binary classification to multiclass problem [50].

For cases where the data is not linearly separable, SVMs can be extended using kernel functions, which map the input features into a higher-dimensional space where a linear hyperplane can effectively separate the classes. Commonly used kernels include the polynomial kernel, the radial basis function (RBF) kernel, and the sigmoid kernel [46], [51].

The basic algorithms derived from SVM for multiclass classification include (1) one-versus-all [52] (2) one versus one [53] (3) directed acyclic graph SVM [54] (4) error-correcting output codes [55].

SVMs have been successfully applied to a wide range of applications due to their robustness and effectiveness in high-dimensional spaces. They are particularly popular in image classification [56], text categorization, bioinformatics [57], and handwriting recognition [36], [58]

2.1.1.4 Hidden Markov Models

Hidden Markov Models (HMMs) are statistical models that describe systems where the observed data $O = \{o_1, o_2, \ldots, o_M\}$ are generated by a sequence of hidden (unobserved) states $S = \{s_1, s_2, \ldots, s_N\}$. Further, it assumes transition matrix

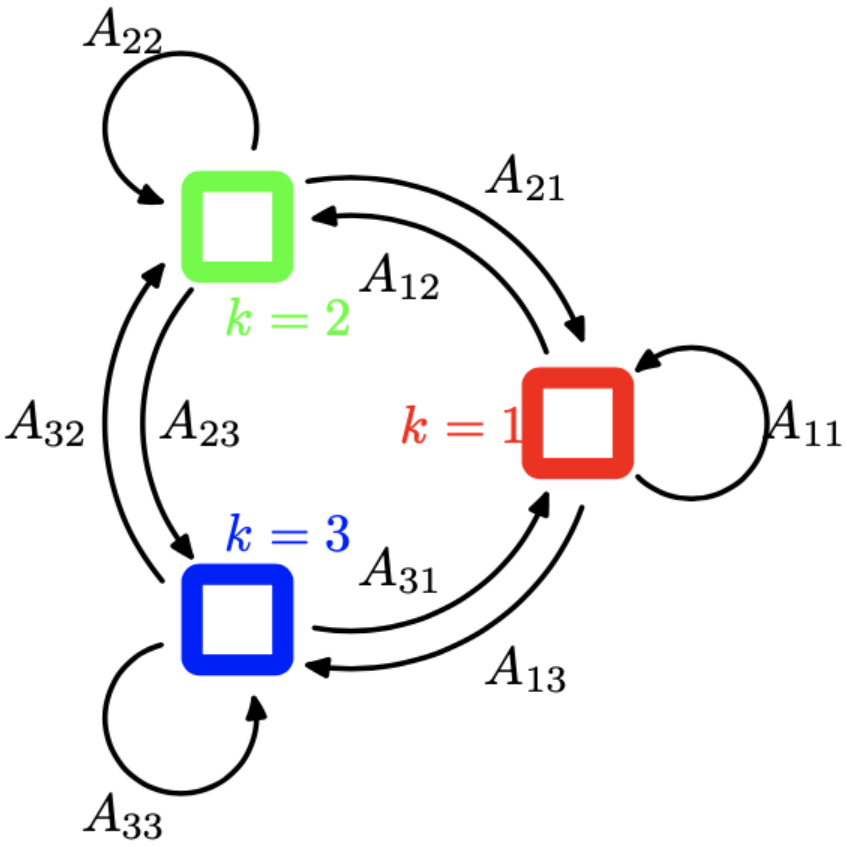


Figure 2.3: An illustration of the transitions between a hidden Markov model with 3 states [40].

$A = [a_{ij}]$ and emission probability $B = [b_j(k)]$:

$$\begin{aligned} a_{ij} &= P(s_j|s_i), \\ b_j(k) &= P(o_k|s_j), \end{aligned} \tag{2.5}$$

where a_{ij} is the probability of hidden state s transits from i to j, while $b_j(k)$ is the probability of observing k given hidden state k. By iteratively maximizing the likelihood, it optimizes the parameters of both the underlying distributions and transition matrix of the Markov system [59], [60].

As the state transitions describes the change of states, the HMM models are widely used in processing time series signals, such as speech recognition [61], [62]

and gesture recognition [63], [64]. To further capture the pattern of data, many variations of HMM models are discussed in the 1990s to 2000s. The high order hidden Markov model is built on the assumption that the current state is dependent on previous states, which grants the model a better ability to analyze time series data [63], [65]. Factorial hidden Markov model extends the basic model structure by integrating several random processes together such that enables the analysis of multiple random variables [61], [66].

2.1.1.5 Decision Tree

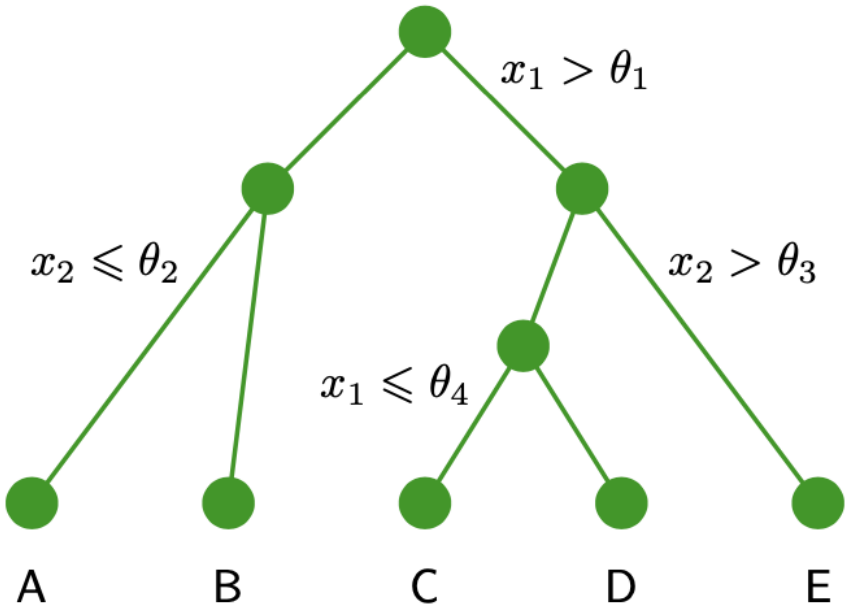


Figure 2.4: An illustration of decision tree [40].

The decision tree adopts a tree-wise structure where each of the node conduct an evaluation on a feature [67]. Based on the evaluation, the example is then categorized into a homogenous subset and further fed into the subsequent node. The training of decision tree is usually based on the information gain, variance reduction or Gini impurity that based on the idea that selecting the feature and criterion by minimizing the entropy or loss of classification results [67]. One of the

famous extension of decision tree is the classification and regression tree (CART) that adopts both discrete and continuous measurement for the nodes [68]. To scale the size of model structure, attempts of integrating multiple decision trees is made to provide a robustness estimation of the observed data, such as random forests [69] and rain forests [70].

2.1.1.6 Pros and Cons

The advantage of statistical models lies in their strong explainability; each model is designed based on certain assumptions and predicts results following specific logic. Due to their high explainability, statistical models are widely adopted in industry for their interpretability, especially in highly sensitive fields such as finance [71], [72], medical diagnosis [73], [74], and automated vehicles [75]. Each of these models presents a detailed analysis of how predictions are made and explains the behavior of the models. This makes it easier to derive insights from the results by tracing back to the factors contributing to the predictions. However, statistical models are also limited by the complexity involved in their construction.

First, task-specific preprocessing is required by statistical models for certain real-world problems. For example, behavior analysis of laboratory animals is an essential part of medical and biological research in verifying the performance and effects of clinical trials [3]. This process requires professional knowledge and is time-intensive. To automate this process, early attempts were made to solve this task with statistical learning methods. For example, the Janelia Automatic Animal Behavior Annotator (JAABA) was introduced to annotate laboratory animals like mice, fruit flies, and larvae [76]. A fly-vs-fly dataset, along with a computer vision-based annotation system, was introduced for analyzing social behavior [77]. The idea of action detection is to encode features of the graph, such as location, orientation, and angles, and classify behavior according to these features using statistical models [78], [79]. However, vision analysis is required to extract features for model fitting and prediction, which reduces the scalability of such models [3].

Second, the complexity of proposed tasks and challenges has dramatically increased in recent years, often surpassing the capabilities of statistical methods. The MNIST handwriting dataset is one of the most popular datasets in the early study of machine learning models [80]. Early works investigated support vector machines (SVM) to perform classification on the MNIST dataset, achieving a best error rate of 3.2% [80]. A later publication introduced a heterogeneous FPGA architecture that uses SVM and boosting accuracy to 98.96% [81]. However, similar results can be easily achieved by a 9-layer fully connected neural network [2]. A relatively more complex dataset is the CIFAR10 dataset, which consists of 60,000 images with 32 $\times$ 32 pixels from 10 categories [82]. Deep learning models have been shown to achieve more than 93% accuracy using deep convolutional networks. Notably, state-of-the-art vision transformer models (ViT) [83], which integrate the idea of transformers [84], achieve over 97% accuracy. On the other hand, there are very few publications based purely on statistical models that achieve comparable results. Instead, some existing works use neural networks as feature extractors and perform classification with K-Nearest Neighbors [85] and support vector machines [86], slightly outperforming the benchmark models. For large-scale image datasets such as ImageNet [87], the most advanced discussions and models are predominantly based on deep learning rather than statistical models.

2.1.2 Neural Networks and Deep Learning

Due to the limitations mentioned above, statistical models have not achieved the widespread usage that deep learning models have in recent years. In contrast, neural networks have shown greater potential in addressing these problems. Inspired by the human brain, artificial neural networks (ANNs) are composed of layers of neurons. Each neuron mimics the behavior of a biological neuron, receiving signals from predecessors, processing the information, and firing another signal to all the neurons in the next layer [88]. The input to the model is received and processed by an input layer, allowing the deep layers to remain unchanged regardless of the shape

of the inputs. Additionally, by adding extra layers and increasing the width of each layer, ANNs have been proven to possess unlimited ability in approximating complex functions [89].

After it was proposed, the concept of artificial neural networks attracted great attention and was popular in the 1980s. However, the performance of neural networks highly depends on their ability to approximate functions with high complexity, which essentially depends on the number of parameters. Moreover, training large-scale models also requires sufficient data to avoid underfitting. Due to limitations in computational resources as well as a lack of data, the popularity of neural networks declined in the 2000s.

In 2010, Nvidia developed the Fermi microarchitecture, which introduced the streaming multiprocessor and GigaThread global scheduler. The streaming multiprocessor consists of 32 CUDA cores optimized for 64-bit and extended precision operations. The GigaThread global scheduler uses a two-level distributed thread scheduler, where the first level schedules thread blocks to all the multiprocessors, and execution is further distributed to all the threads at the streaming multiprocessor level. This innovation improved the computational ability of GPUs and is still in use today. Simultaneously, the development of the Internet industry and advancements in hardware made collecting, labeling, and storing vision or text data easier, providing a solid foundation for training large-scale models. In particular, challenges such as COCO [90] and the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [87] were proposed in the 2010s and greatly encouraged research in deep learning.

Driven by advances in computational power and the availability of data resources, research in machine learning has made astonishing progress in the past decade. In fact, large-scale models have already become one of the general solutions in various fields, such as computer vision, natural language processing, and generative models. Research in computer vision started with the recognition of single objects. The most prominent models, such as VGG networks [91], ResNets [14], and GoogLeNet [92],

have since been adopted as backbone feature extractors in the research of object detection in large-scale images. Similar research has also extended to videos, where the data becomes 3D with the inclusion of the time dimension [93], [94].

Another important research thread is natural language processing (NLP), which aims to construct models that perform tasks on language that can be completed by humans, such as language translation [95], [96], quiz answering [97], [98], and sentiment analysis [99], [100]. Other topics include generative models that aim to create images or texts based on given descriptions, and descriptive models that build connections between language and models [101]–[103].

2.1.3 Large Scale Models and Its Risks

Under such background, new models are consistently being introduced with better performance and generalization. One of the most inspiring recent innovations is the launch of ChatGPT. As its name suggests, ChatGPT is a chatbot developed by OpenAI. Unlike its predecessors, ChatGPT can accurately answer questions based on prompts across various fields and continue the same conversation with memory of previous information. Additionally, it integrates generative models, enabling it to generate images based on descriptions. Powered by ChatGPT, AI assistants have been introduced by fine-tuning the model for particular tasks, such as creating PowerPoint presentations, writing code based on annotations, and more. The potential of ChatGPT is still being explored [104]. Simultaneously, discussions about ChatGPT have emerged in academia [105], with several recent works investigating its use across finance [106], public health [107], and even its impact on academia itself.

One significant concern in these discussions is the risks associated with ChatGPT. In some cases, ChatGPT has been reported to generate fabricated information, such as fake literature, coherent but biased and incorrect content, and even malicious or harmful responses when prompted with elaborately engineered inputs [108]. The regulation of large language models is proactively being discussed by OpenAI

and governments [109]. However, the astonishing performance of ChatGPT is backed by 100 billion parameters, making it difficult to explain why and how it responds to certain prompts, and even harder to control the associated risks. In this context, research on the interpretability of large-scale models has become increasingly important [110].

2.2 Feedforward Neural Networks and Explainability

The objective of this dissertation is to shed lights on the explainability of feedforward neural network (FNNs) from activation function perspective. Therefore, it is necessary to:

- describe the feedforward neural networks and their differences with recurrent neural networks,
- list the main playing fields of such type of FNNs,
- review the existing investigation of explainability of FNNs,
- discuss the development in activation function of neural network.

This section provides literature reviews of above topics.

2.2.1 Feedforward Neural Networks and Recurrent Neural Networks

Feedforward Neural Networks (FNNs) represent one of the earliest and most fundamental types of artificial neural networks. FNNs are structured such that data flows in a single direction, from the input layer through any hidden layers to the output layer, with no cycles or loops. This makes them well-suited for tasks that inputs are independent of each other, such as image classification and regression.

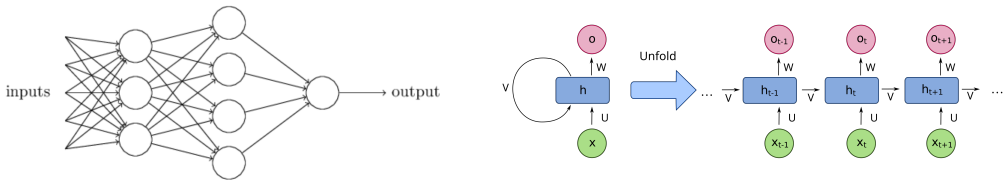


Figure 2.5: (left) An illustration of 3 layers feedforward neural network; right: An illustration of recurrent neural network [40].

The concept of FNNs was introduced with the development of the perceptron which demonstrates the ability of a simple model to perform binary classification tasks [111]. However, the initial enthusiasm was tempered by the realization that single-layer perceptrons could not solve non-linear problems[112]. The introduction of Multi-Layer Perceptrons (MLPs) addressed these limitations by incorporating one or more hidden layers, allowing FNNs to model complex, non-linear relationships. The proposal of back-propagation algorithm is a significant breakthrough that enables efficient training of these deeper networks by computing gradients for each layer [113]. This advancement propelled the adoption of FNNs in various applications, particularly in image and speech recognition.

Due to the limitation of datasets and computation power, its fashion fell out in the 2000s. Recent years, with the emergence of large annotated datasets and the development of high performance computing techniques, deep models were proven to be effective by many proposed models. In particular, the sophisticated architectures of feedforward neural networks are achieving significant importance in accuracy and demonstrating the practical utility of neural networks in various fields, including object detection, action recognition and medical diagnosis [114].

Deep feedforward neural networks do not have memory while they treat each input independently and cannot retain information from previous inputs, therefore not suitable for processing sequential data. This is instead addressed by introducing loops within the network [115]. The introduction of Long Short-Term Memory (LSTM) networks in 1997 addressed the vanishing gradient problem that plagued

traditional RNNs, enabling the training of deeper and more powerful networks [116]. LSTM networks include mechanisms such as forget gates and memory cells, which allow the network to retain important information over long sequences. Gated Recurrent Units (GRUs) further simplified the LSTM architecture while maintaining similar performance [117].

RNNs have found famous applications in various domains. In natural language processing (NLP), they have been used for tasks like machine translation, as demonstrated by the success of sequence-to-sequence models with attention mechanisms, which significantly improved the performance of translation systems [118]. In speech recognition, RNNs and their variants have been integral to advancements in automatic speech recognition systems, enabling more accurate transcription of spoken language [119]. Additionally, RNNs have been applied in time series prediction tasks, such as financial forecasting and climate modeling, where understanding the temporal dynamics of data is crucial.

2.2.2 FNN Applications

As discussed above, feed forward neural networks are widely applied in tasks where inputs are independent, such as image related tasks. In this section, we briefly discuss several such topics where feed forward neural networks are achieving state-of-art results.

2.2.2.1 Image recognition

One of the main playing field of feed forward neural network is image recognition. The objective of image recognition is to identify the category of the object in an image. As the cornerstone topic in the field of computer vision, there are many widely recognized large scale datasets, including ILSVR [120], PASCAL VOC [121], COCO [90], CalTech-101 [122]. Among them, the ImageNet is the large scale object recognition challenge that inspired a great amount of backbone networks for learning the representations and extracting features.

The first popular network is the AlexNet that competed ImageNet challenge on 2012 with a 15.3% top-5 error rate [123], which outperforms the second-best entry by 10%. AlexNet uses convolutional layers, max-pooling layers and fully connected layers with more than 60 million parameters, which opens the gate of deep learning models. Following AlexNet, VGG nets further increases the depth of model with 11 to 19 layers on different structures [91]. Although it is shown that deeper networks have better performance compared with the shallow networks, the boost is not significant and the training is much more difficult. The residual network (ResNet) aims to address ImageNet problem by introducing short-cut layers, which grants the network the ability to converge for deep structure [14]. ResNext leverage the idea of residual block and aggregates a set of convolutional layers from different paths with same topology and improves the performance of ResNet [124]. The DenseNet introduces a dense connection between layers but also preserver the feed-forward of neural network, which alleviate the gradient vanishing issue and encourages use of the feature [125]. The InceptionNet is one of the most advanced backbone structure that combines the advantages of previous works by adopting kernels at different sizes [92], [126]. To improve the portability, MobileNet balance the trade-off between model performance and scale by introducing mobile models with reduced size [127]–[129].

2.2.2.2 Object detection

Object detection is more advanced task on top of image recognition, which requires identifying both the categories and locations of all objects in an image. For the large scale image object detection task, R-CNN [130] introduced an inspiring two-stage architecture by combining a proposal detector and region-wise classifier. SPP-Net [131] and Fast R-CNN [132] are then introduced with the idea of region-wise feature extraction which significantly speeds up the overall detector. Faster-RCNN [133] proposed a Regional Proposal Network, which is almost cost free by sharing convolutional features with detection network, for object bounds prediction. A

multistage detector Cascade R-CNN [134] is then proposed which improves the accuracy of detection by setting increasing IoU thresholds for a sequence of detectors.

One stage object detection, as an alternative architecture, is also popular due to their computational efficiency. YOLO [135] implemented with an efficient backbone network and enables real time object detection. Single shot multi-box detector (SSD) [136] uses multiple feature maps at multiple resolution to cover objects with different scales and detects objects similarly to Region Proposal Network (RPN) [132]. The cost of high computation speed of one-stage detector is that their accuracies are below the most two-stage architectures. However, RetinaNet [137] achieved a better result than most two-stage objects detectors by addressing foreground-background imbalance in dense object detection.

2.2.2.3 Medical Diagnosis

One practical application derived from the image related tasks is the medical diagnosis. By leveraging the ability of extracting and understanding features of feedforward neural networks, several landmark studies demonstrating the effectiveness of these models in various medical tasks. [114] developed a deep learning algorithm for detecting diabetic retinopathy in retinal fundus photographs, achieving performance comparable to that of ophthalmologists. Similarly, [138] introduces CheXNet, a deep learning model that detects pneumonia from chest X-rays with radiologist-level accuracy, setting a new standard for medical image analysis. [139] applies deep convolutional neural networks to classify skin lesions, achieving dermatologist-level classification performance, highlighting the potential for AI to assist in dermatological diagnostics. These studies underscore the transformative potential of deep learning in improving diagnostic accuracy and efficiency in healthcare.

2.2.3 Explainability of Neural Network

Since the beginning of the deep learning era, large-scale models have been criticized for their non-transparency and lack of explainability [140]. They are often viewed

as black boxes that take inputs and produce outputs without providing information about the decision-making process. Explaining and reasoning about these black boxes is a long-standing topic in deep learning research. The goal of investigating the explainability of deep learning models is to enhance their success, mitigate the limitations of opacity, and control the risks of potential malicious outputs.

There are different threads of literature regarding the explainability of deep learning models. In general, most related research aims to understand certain observations, provide explanations for those observations, and potentially introduce solutions to improve model performance from different aspects.

The first topic is understanding the potential of deep neural networks. Due to the scalability of neural networks, deep models with millions of parameters have better potential in approximating highly complex objective functions, which grants them the ability to solve complicated problems. One of the most prominent arguments in this thread is the *universal approximation theorem*, which states that given an arbitrary objective function, it can be approximated by a neural network with a certain depth or width to arbitrary precision [15], [28]. Starting with the arbitrary width case, subsequent works have verified this theorem in various forms, including the arbitrary depth and bounded width case [141], bounded depth and bounded width case [142], limited depth case [143], CNN architecture [144], and graph networks [145].

One of the most commonly asked questions in this field is why deep learning models outperform traditional statistical models and what the potential of deep learning models is. Early attempts at explaining performance started from a statistical perspective. The generalized additive models (GAM) use smooth functions to analyze the behavior of neural networks [146]. TREPAN is a decision-tree-based method that aims to produce a comprehensible concept description while maintaining the same classification results as the network [147]. Another approach is to study the conditional expectation of features from the neural network and explain the results using partial dependence plots, which present the marginal effects

of features [148], [149].

With the development of deep learning, it has become difficult for statistical models to explain the features of highly complex neural networks. One straightforward way to understand deep convolutional networks is to visualize the features extracted by the network from inputs. This was achieved by the proposal of DeConvNet, which uses deconvolutional layers and up-pooling to reverse the computation of the neural network [150]. The results of DeConvNet show how each filter in the convolutional layer extracts features from the input layer. However, the accuracy of input reconstruction is limited by the max-pooling layer. To address this issue, later works introduced neural networks with fully convolutional layers, replacing max-pooling with larger strides in the convolutional layers [151]. To further understand how predictions are made by the network and how they are affected by perturbations, subsequent works studied the features of texture [152], color [153], attention area [154], and partial images [155] by visualizing the intermediate layers.

As it provides a theoretical guarantee of the output-to-input change ratio, the study of Lipschitz continuity has a long history in neural network literature [156]. However, calculating the Lipschitz constant is an NP-hard problem even for a two-layer multilayer perceptron [157]. To estimate the Lipschitz constant of a neural network, various methods have been proposed, including the use of regularization in kernel methods [158], PAC-Bayes theory [159], and others [160]–[163]. On the other hand, some theoretical works focus on describing the generalization ability and robustness of neural networks with the help of Lipschitz continuity [159], [164], [165]. Several works also contribute to this area by investigating the norms of network components [166], [167].

One strand of research related to this work stems from the intuitive observation that neural networks with piecewise linear activation functions also provide linear mappings from a domain to a range [168]. Each of these piecewise linear domains is referred to as a *linear region* [169]. By identifying the activation status of each neuron, it is possible to explore the expressive ability [19], limitations [170], and

explain the model performance [2]. On the application side, several recent works utilize the linear properties of ReLU networks to bound the network's l_p norm [171] and search for adversarial examples [172].

2.2.4 Activation Function and Its Development

The investigation of activation functions is a long-standing topic in deep learning research [173]. Earlier studies of neural network limitations have documented issues such as vanishing and exploding gradients [17], [174]–[176], the instability of network predictions [177]–[179], and the limitations of deep model expressive ability [180], [181]. Other approaches aim to explain neural networks by exploring the approximation function of the network, including sensitivity [177]–[179], complexity [182]–[184], and the theoretical guarantees of model expressive ability [15], [141], [185]–[187].

On a broader scope, instead of focusing solely on explaining existing architectures, there are approaches that deal with explainable-by-design machine learning methods. To better align input features, some works [188], [189] suggest replacing the linear mapping of networks with non-linear operators. Inspired by the human recognition system, several works aim to quantify and prototype visual input according to basic semantic units to integrate interpretability into the model structure [190], [191]. Other works seek to improve the performance of deep models by simulating the recognition process [190], [191]. In a more general sense, these works are referred to as prototype-based classifiers, which rely on the similarity of data to a given prototype rather than pure evaluation metrics [192].

2.3 Related Downstream Applications

This section reviews two downstream applications of feed forward neural networks that related to this dissertation. Based on the investigaiton of model expressive ability and model robustness, Chapter 5 and Chapter 6 introduce a pruning algorithm

named *Neuron Entropy Pruning* (NEP) and a randomized smoothing algorithms named *Smoothed Classifier with Reformed Float Path in dual direction* (SCRFP-2), respectively. This section discusses the neural network pruning algorithms and robustness training algorithms that related to the proposed algorithm.

2.3.1 Network Pruning

Network pruning aims to remove unnecessary parameters to reduce the scale of a neural network with minimal compromise to model performance [193]. Since the proposition of this idea in the 1990s [194], [195], research has explored three mainstream approaches: neural architecture search (NAS), weight optimization, and magnitude-based pruning.

Neural architecture search, as suggested by its name, aims to find an optimal structure from a well-trained model given a certain resource budget [196]. It involves training a mega-network, sampling subnetworks from the weights, and fine-tuning the sampled networks [197]. Different designs of NAS algorithms essentially represent trade-offs between the time spent on various steps, including training super-networks [198], [199], optimizing network structure with reinforcement learning [200], [201], and using gradient-based methods [202], [203]. Weight optimization, on the other hand, focuses on compressing the weights of each layer using various dimension reduction methods so that the outputs of layers are minimally affected [204], [205].

Compared with the above approaches, magnitude-based pruning methods are faster and more intuitive while still delivering competitive performance with complex models. Based on the idea of removing parameters with little contribution to prediction, weight-based pruning eliminates the smallest weights in models [206]–[208]. Subsequent works have introduced gradient-based [209] and feature-tracking methods [210] to assign importance scores to parameters.

2.3.2 Model Robustness

While deep learning methods are achieving state-of-art performance in various fields, recent work shows that most of the well-trained models are vulnerable to maliciously designed inputs, which referred to as adversarial examples[177], [211], [212]. To explain the existence of adversarial example, previous works presented several hypotheses, including *linearity hypothesis* and its variants [178] and *Evolutionary stalling*[179].

After that, the Fast gradient sign method (FGSM) set the fundamental of white-box adversarial attack methods by adding a bounded noise whose direction is the same as the gradient of the cost function with respect to the input and provide a simple solution to generate stable adversarial example with linear perturbation [213]. RFGSM [214] then enhanced it by applying a small random step before calculating the image gradient, therefore allowing a non-linear perturbation. FFGSM then suggest increasing the non-linearity of the perturbation by replacing the fixed-length step with a random initialization with the boundary [215]. In particular, when FFGSM adversarial examples are adopted in training, the model is able to achieve comparable robustness with models trained with multistep adversarial.

As the objective of adversarial is finding a constraint perturbation that maximizes the loss of the model given input and current parameters of the model, it can be viewed as an optimization problem. Based on the thought, the basic iterative method (BIM) perform the search of local maximum by taking multiple, smaller steps, and therefore achieve a better success rate in attacking [216]. The method is then improved by introducing multiple random restarts, and known as projected gradient descent (PGD) adversarial. Since the multiple restarts and multiple steps in each restart ensure a stable performance of PGD attack in the local optimization problem, PGD adversarial is widely accepted and regarded as the strongest attack. Many following works are then proposed based on it. The momentum iterative fast gradient sign method integrating (MIFGSM) the momentum term into the iterative process for the attack to boost the adversarial attack [217]. Auto-PGD (APGD)

introduce an automatic scheme for step size selection, and PGD on Difference Logits Ratio (PGDDLR) method using a proposed DLR loss to avoid potential failure of PGD attack [218]. The author also introduces an auto-attack method that is parameter-free and able to achieve a comparable attack success rate with other state-of-art methods. Another notable attack is the CW attack which crafts adversarial examples by tailoring to three distance metrics and optimizing the loss with gradient descent. Because of its outstanding performance, it is always used to test the robustness of neural networks.

Many solutions are proposed apart from adversarial training in response to the threat models. Detection methods introduce a small subnetwork to detect the perturbation[219], [220]. Defensive distillation uses a distillation network to learn the output function of the previous network and predicts the class probabilities from the first network[221]. Denoiser scheme adopts a high-level representation guided denoiser as a defense to remove the perturbation added by the adversarial models[222]. Most of the forehead mentioned methods are efficient against previous attacks but proved to be vulnerable against the latest threat models[223][224][221]. The ongoing battle between attack methods and defense methods suggest that it is hard to evaluate the robustness of defense models as well as the threat models.

On the other hand, adversarial training, as the first proposed solution against adversarial examples, remains popular due to its explainability. In particular, the PGD adversarial training is still argued to be the strongest defense[225], [226]. However, since the back propagation and forward pass are calculate multiple times for a single batch, PGD adversarial training is computationally expensive. Many works are proposed to address the issue[225], [227]–[229]. Intuitively, to reduce the computational cost, the algorithm should perform fewer backwards and forward pass. FreeAT eliminates the additional cost of generating PGD adversarial by recycling the gradient information to update the model weight, so the model is updated same steps with fewer epochs, achieving a comparable result to PGD training[227]. You Only Propagate Once (YOPO) suggest that adversary update is only coupled

with the parameters of the first layer of the network, freezing the forward and back propagation within the first layer can greatly reduce the training time[228]. Moreover, fast adversarial training shows that the model can be trained with FGSM with a random initialization using a DAWNBench training schedule and achieve comparable results with less time.

Another finding in the research of FFGSM attack is the catastrophic overfitting. The catastrophic overfitting refers to the phenomenon that, during the single-step adversarial training, the model accuracy against PGD attack suddenly decrease to 0% in a few epochs, while the accuracy against natural images as well as FGSM adversarial increase dramatically. Later, it is found that it also happens to other single-step training methods[230]. Although some methods are proposed to address the issue[230], [231], those methods are computationally inefficient and fail to achieve the same robustness level as the non-overfitted FFGSM training.

Early works on improving the robustness of neuron networks focused on adversarial training methods [214], [217], [225], [232], [233], while recent investigation shows adversarial training methods can always be broken by more advanced attacks [234]. Certifiable training is an emerging solution to the above issue, which aims to provide a certified region, within which the input data are free from an attack. By viewing the training as a convex optimization problem, dual relaxation approaches apply duality to provide a sound bound for training as well as verifying the network [235], [236]. An alternative is to estimate the Lipschitz boundary of the network and introduce constraints on either objective loss [160] or forward propagation [237]–[240]. However, verifiable training comes with a compromise on accuracy and scalability.

On the other hand, randomized smoothing is a randomized algorithm that applied to the base model, therefore is able to provide robustness without affecting performance of base classifier. [241], [242] first propose to ensemble the information around input data to smooth the prediction, but fail to provide a theoretical guarantee on the result. Randomized smoothing and related works provides a

theoretical analysis of the certifiable with Monte Carlo methods [241], [243], [244]. Following works further applies consistency regularized [245], convex combination of samples [246] and adversarial attacks [247] to further increase the robustness accuracy.

Chapter 3

Activation Function and Model Performance

As discussed in the introduction, there is a noticeable performance gap between neural networks with different activation functions under the same structure. This chapter aims to verify the observation by comparing the performance of neural networks with different activation functions on several benchmark models. Further, it investigates the weights and gradients of those neural networks with different activation functions with an objective of understanding the effects of proposed techniques. It is found that the neural networks with saturated actiavtion functions are able to coverage during training but their performance worse than ReLU networks. At last, this chapter provides a theoretical analysis to understand how current weights affect the training for neural networks with different activation functions and shows that the weights of symmetry activation functions are more likely to stuck at local minimum despite having valid gradient. This suggests the additional investigaiton is necessary to understand the role of activaion functions and how they can affect the model performance.

The structure of this chapter is listed as following. Section 3.1 aims to validate previous literatures and examine whether there still exists the performance diffferences between neural networks with Sigmoid and ReLU activation function.

As an introductory experiment, the objective in this section is to verify that whether the Signomid and Tanh neural networks are trainable after introducing batch normalization and weight initialization. Further, we compare the performance between neural networks with different activation functions to understand the

Section 3.2 analyze the weight and gradient distributions during training to check whether *vanishing gradient problem* is addressed by the proposed techniques. One of the most famous cause of the bad performance of neural networks with saturated functions is the *vanishing gradient proble*, which suggests that the derivative of such activation functions is relatively small when the inputs are located within the saturated region. This analysis reveals that, during the training, neural networks with saturated activation functions have valid gradient after introducing normalization of weights and gradients.

Section 3.3 explains the problem from a theoretical perspective by formulating the relationship between model weights and gradients. It is found that during backpropagation, the gradient not only depends on the current weight but also heavily relies on a transformation factor determined by the form of the activation function. This factor affects the learning efficiency and accuracy of Sigmoid and ReLU networks differently.

To further illustrate the findings, Section 3.4 describes the learning dynamics of neural networks with different activation functions using a toy model. It is found that for networks with saturated activation functions, some neurons fire similar outputs regardless of the inputs and fail to update their weights during training. This phenomenon is referred to as the *dying neuron issue*.

3.1 Model Performance and Activation Functions

This work is motivated by the interest in understanding the performance gap caused by different activation functions. In particular, the introductory question concerns whether the above techniques have fully addressed the *vanishing gradient* problem.

The research undertaken in this section comprises several steps:

- First, a descriptive discussion of illustrative datasets and dthe activation functions and their properties is presented to justify the choice of activation functions studied in this section.

- Second, to demonstrate the existence of a performance gap, a comparison of the performance of models with different activation functions is presented. Batch normalization and weight initialization are separately introduced to the model with the objective of investigating their effects on regulating weight distribution and improving model performance.

- Finally, it presents the weights, gradients, and weight-to-gradient ratios of different models across layers, with the objective of verifying whether the gradient indeed vanishes during backpropagation.

3.1.1 Datasets

As an introductory investigation, this Chapter leverages the MNIST dataset and CIFAR10 dataset to verify the performance difference between neural networks with different activation functions. The MNIST and CIFAR-10 datasets are two of the most widely used benchmarks in the field of machine learning, particularly for evaluating the performance of image recognition algorithms. Each serves as a fundamental resource for developing and testing new machine learning models, but they target different aspects of visual recognition.

The MNIST (Modified National Institute of Standards and Technology) dataset is a collection of 70,000 grayscale images of handwritten digits (0-9). Each image is 28x28 pixels in size. It is divided into 60,000 training images and 10,000 testing images. MNIST is primarily used as an entry-level dataset for machine learning and computer vision, serving to benchmark and test classification algorithms. The simplicity of the images makes MNIST manageable for testing basic and complex

neural networks. Its widespread use has established it as a baseline for comparing model performance in digit recognition tasks.

In contrast, the CIFAR-10 dataset comprises 60,000 colored images (32x32 pixels) spread across 10 different classes such as airplanes, birds and ships (6000 images per class). Similar to MNIST, CIFAR-10 is also divided into 50,000 training images and 10,000 test images. The primary challenge of CIFAR-10 lies in its complexity relative to MNIST due to color information and the diversity in object classes. This dataset is used to advance the development of more sophisticated algorithms that can handle complex visual data and perform accurate classification across a range of conditions and object types.

3.1.2 Activation Functions and Their Properties

The first stage of this empirical discussion is to validate that models with different activation functions exhibit varied performance. Although this argument is widely recognized, a purposeful comparison of the effects of activation functions is helpful in identifying potential issues and explaining the performance gap.

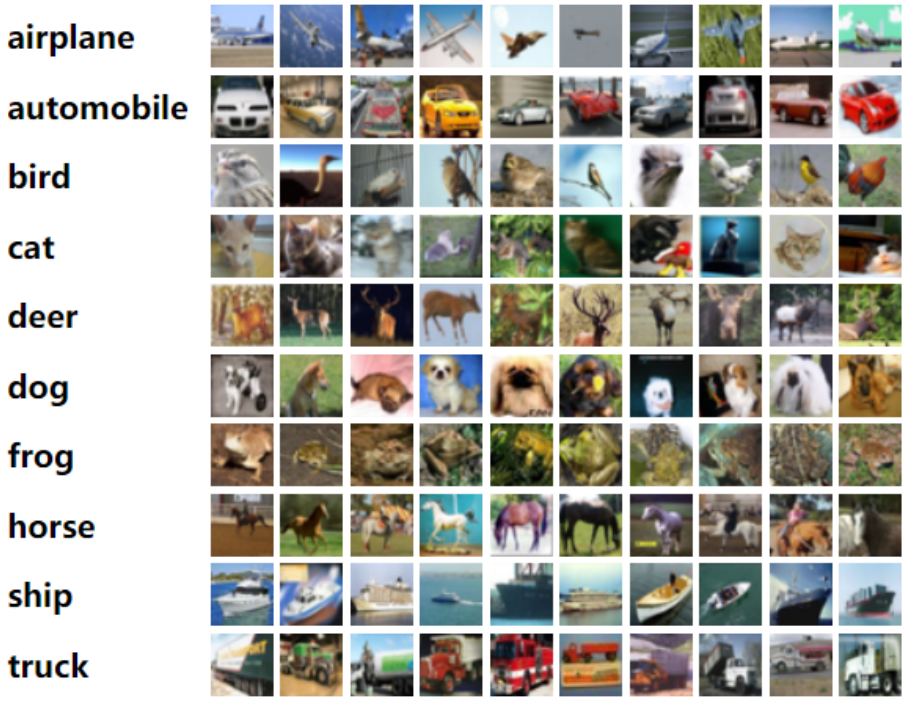
As the objective of the introductory questions is to explain the effect of activation functions on model performance, a discussion of the activation functions and their properties would be beneficial for understanding the topic.

	Sigmoid	tanH	ReLU	GeLU	SeLU
saturated	✓	✓	✗	✗	✗
symmetric	✓	✓	✗	✗	✗
linear	✗	✗	✓	✗	✗
monotonic	✓	✓	✓	✗	✓

Table 3.1: Properties of activation functions.

Table 3.1 shows common activation functions as well as their properties that might affect the behavior of the neural network, including whether they are

(a) Graph of Activation Functions



(b) Gradients of Activation Functions

Figure 3.1: Example of MNIST and CIFAR-10 dataset.

saturated, symmetric, linear, or monotonic. The definitions of these properties are:

- *Saturated*: A function π is right saturated if the limit of its derivative $\pi'(x)$ approaches zero as $x \to \infty$. The function is left saturated if $\pi'(x)$ approaches

zero as $x \to -\infty$.

- *Symmetric*: A function π is said to be symmetric at 0 if $\pi(x) = \pi(-x)$ for any $x \in \mathbb{R}$.

- *Piecewise Linear*: A function π is piecewise linear if its domain can be separated into several regions $R_1, \ldots, R_n$ such that $f(x+y) = f(x)+f(y)$ and $f(x) = \frac{x}{y} f(y)$ if $x, y \in R_i$.

- *Continuous*: A function is said to be continuous if $\lim_{x \to c} \pi(x) = \pi(c)$ for any $c \in \mathbb{R}$.

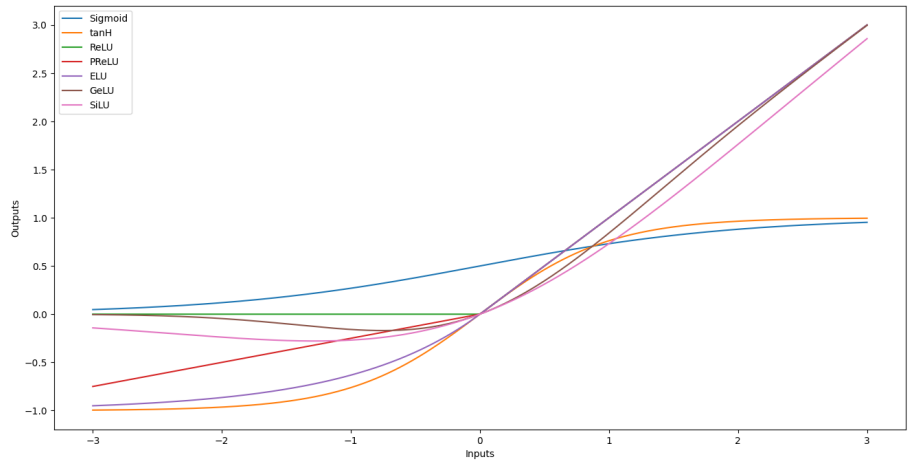
- *Monotonic*: A function is said to be monotonically increasing if for all $x, y \in \mathbb{R}$ such that $x \leq y$, $\pi(x) \leq \pi(y)$.

Figure 3.2 presents the graphs of the above activation functions as well as their derivatives. The choice of activation functions to be compared is critical to the research results. To cover a wider range of function properties, the functions selected in this section are: Sigmoid, Tanh, ReLU, GeLU, and SeLU:

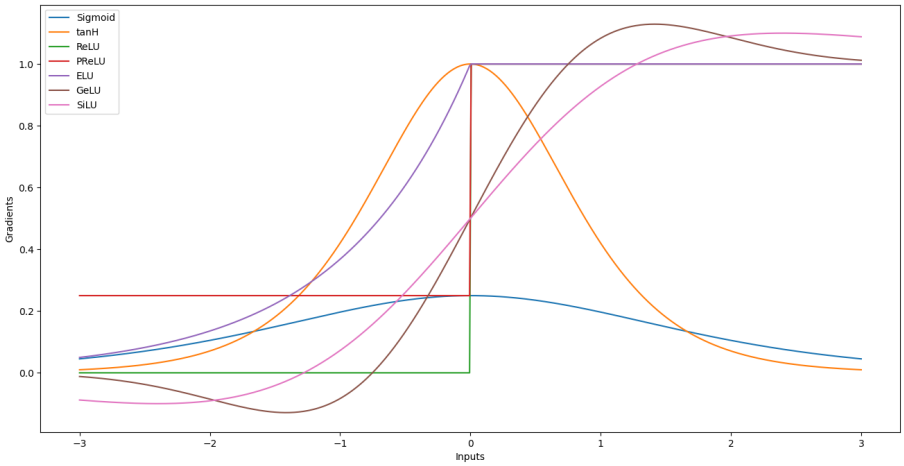
- Sigmoid and Tanh have similar properties, both being saturated at extreme values and symmetric around 0. However, Tanh has a wider range compared to the Sigmoid function.

- ReLU is a piecewise linear activation. Since piecewise linear activation functions share similar properties, PReLU, ReLU6, and other similar activations are not included in the comparison.

- GeLU is the only non-monotonic activation function in this selection. GeLU is left saturated at 0, but its minimum is reached at around -0.8, meaning that GeLU decreases in the interval $(-\infty, -0.8)$ and then increases like other activation functions.



(a) Graph of Activation Functions



(b) Gradients of Activation Functions

Figure 3.2: Graphs of Activation Functions and Their Derivative.

3.1.3 Model Performance

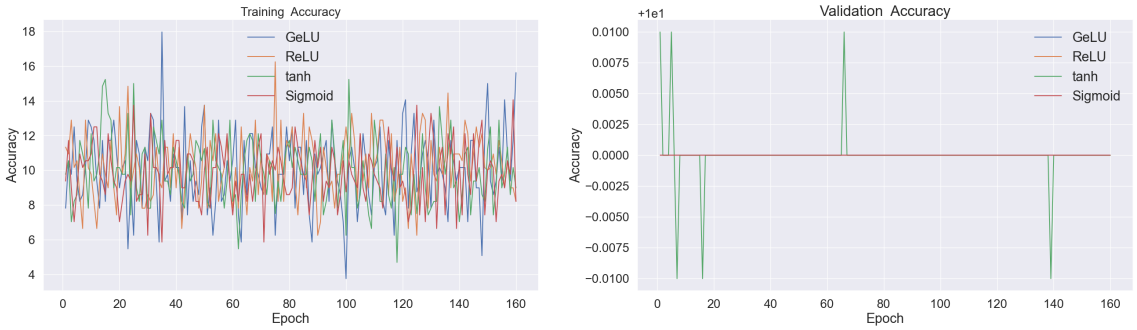
The next step of this investigation is to verify the effects of activation functions on model performance and answer the question of whether the *vanishing gradient* issue is resolved by the proposed methods. The set of experiments in this section compares the performance of deep fully connected networks (DNN), VGG16, and ResNet34 to cover different network structures: linear layers, convolutional layers, and bottleneck blocks.

The VGG16 and ResNet34 models are trained on the CIFAR10 dataset for 160

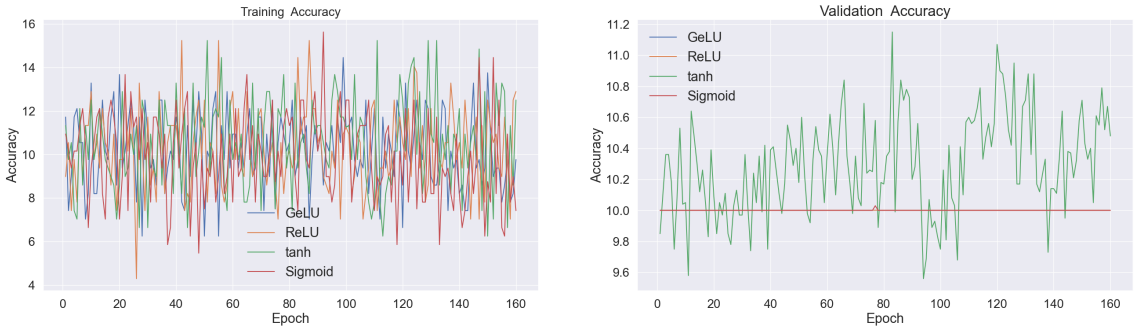
epochs using the SGD optimizer with a batch size of 256. The learning rate is initially set to 0.1 and decays by a factor of 10 after 80 and 120 epochs, following a milestone learning rate schedule. To mitigate the effects of initialization and early training examples, the networks undergo a warm-up phase with a learning rate of 0.001 for 500 iterations.

To demonstrate how batch normalization and weight initialization affect model performance, each model is trained both with and without these methods, resulting in four sets of experiments for each model.

3.1.3.1 Benchmark Models



(a) VGG16 Model



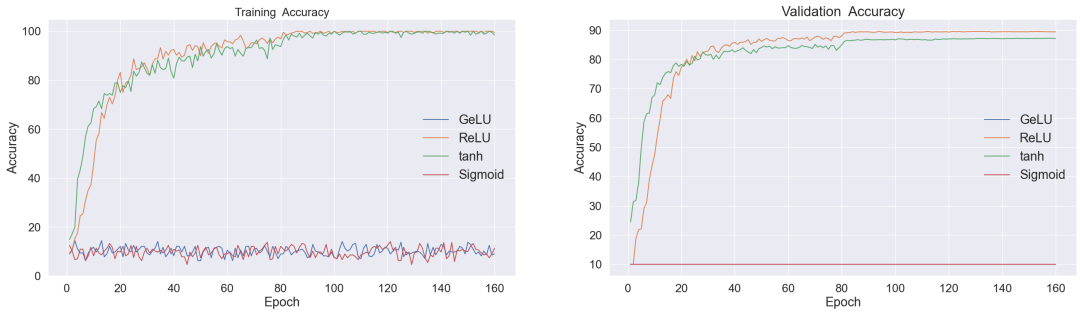
(b) ResNet34 Model

Figure 3.3: Trainning and Validation Accuracy of VGG16 network and ResNet34 Network without batch normalization or weight initialization.

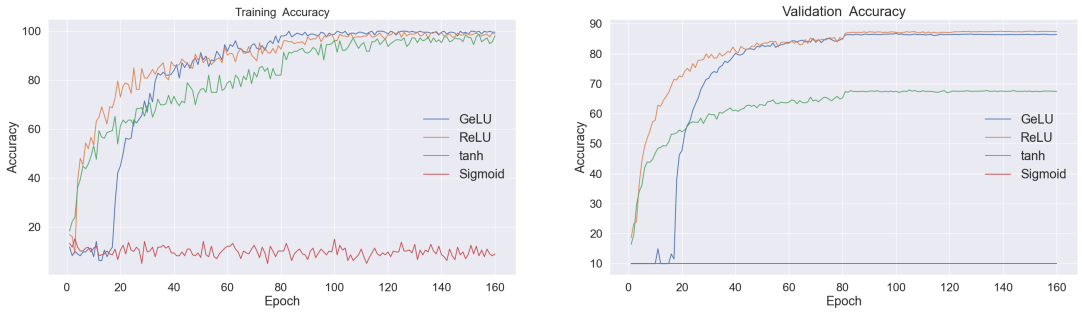
Figure 3.3 presents the benchmark training and validation accuracy of the

VGG16 and ResNet34 models. During training, both the training accuracy and validation accuracy oscillate around 10%. The results show that for deep learning models without batch normalization or weight initialization, training fails to converge regardless of the activation function.

3.1.3.2 Benchmark + Weight Initialization



(a) VGG16 Model



(b) ResNet34 Model

Figure 3.4: Trainning and Validation Accuracy of VGG16 network and ResNet34 Network with weight initialization but without batch normalization.

The second set of experiments aims to illustrate the effect of weight initialization on the performance of deep learning models. After the discovery of the vanishing gradient issue, researchers sought to understand why standard gradient descent was performing poorly and to find a way to enable backpropagation in deep networks without losing the gradient.

Weight initialization was then proposed with the objective of preventing the pre-activation values of neurons from clustering in the saturated region, as it was found

that it is difficult for saturated functions, such as Sigmoid and Tanh, to move out of saturation. To achieve this, the weights of a layer are initialized according to the structure of the layers, ensuring that the gradients of the neural network are normally distributed around 0 to avoid vanishing or exploding during backpropagation.

For example, normalized Xavier weight initialization generates the weights for a fully connected layer with a distribution [17]:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n+m}}, \frac{\sqrt{6}}{\sqrt{n+m}},\right] \tag{3.1}$$

where U is a uniform distribution, and n and m are the sizes of the inputs and outputs for this layer, respectively. Similarly, He weight initialization for ReLU activation functions aims to produce a weight distribution that avoids the pre-activation values of ReLU clustering in the negative domain [18]:

$$W \sim N(0, \sqrt{2/n}), \tag{3.2}$$

where $N(\cdot, \cdot)$ is a normal distribution, and n is the number of inputs for the layer.

Figure 3.4 compares the training and validation accuracy of models with different activation functions after introducing weight initialization techniques. The ReLU network shows the best performance, as both models converge on the training dataset without a noticeable performance gap with the validation dataset. The Tanh network, on the other hand, also converges on the training datasets but only achieves around 60% validation accuracy. The Sigmoid networks perform the worst, as both models again fail to converge, even with weight initialization.

An interesting result is observed with the GeLU networks. The GeLU activation function is known as a potential improvement over ReLU, as it has both non-linear and non-monotonic regions but shares similar properties with ReLU. However, the VGG16 model with GeLU fails to converge. This suggests that despite GeLU's ability to outperform ReLU in certain cases, it may not be as robust as the widely adopted ReLU activation.

3.1.3.3 Benchmark + Batch normalization

Another effective correction for the vanishing gradient problem is the batch normalization layer. The objective of batch normalization is to re-center and re-scale the output of the linear affine transformation ϕ so that the intermediate features have a zero mean and unit variance before being passed to the activation function.

Given an input $x^{(i)}$ at layer i, the linear affine transformation maps the input to $W^{(i)}x^{(i)}$. A standard batch normalization layer records the mean $\mu^{(i)}$ and variance $\gamma^{(i)}$ of the historical data to estimate the distribution. After rescaling and resizing the data to a distribution with zero mean and unit variance, a linear transformation is then applied to the data using learnable parameters $\gamma^{(i)}$ and $\beta^{(i)}$:

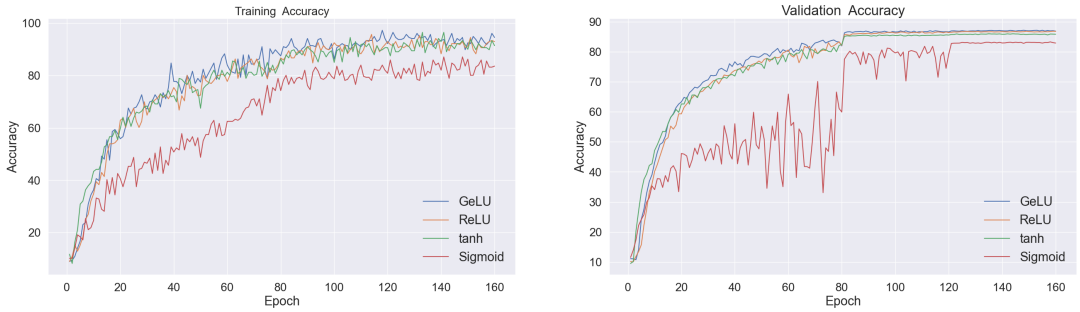
$$z^{(i)}(\mathbf{x}) = \frac{W^{(i)}x^{(i)}(\mathbf{x})^T - \mu^{(i)}}{\sigma^{(i)} + \epsilon}\gamma^{(i)} + \beta^{(i)}, \tag{3.3}$$

where $z^{(i)}$ is the pre-activation value of layer i.

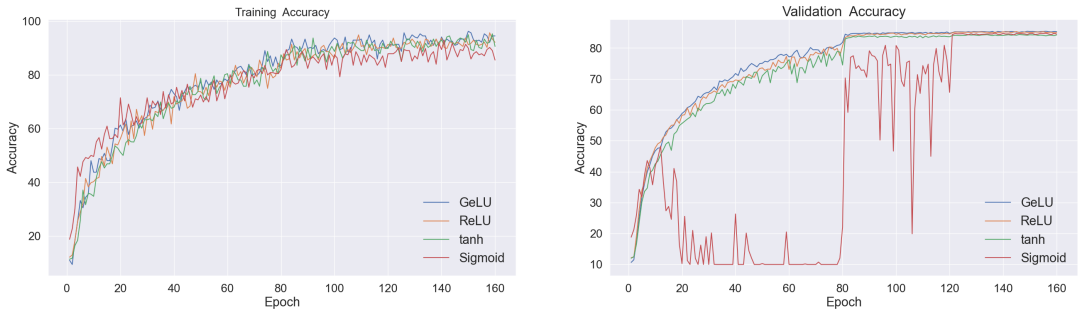
Based on this idea, other normalization techniques have been proposed, such as layer normalization [248], group normalization [249], and weight normalization [250]. To better accommodate different model structures, re-centering and re-scaling can be applied to different dimensions, including layer normalization, instance normalization, and group normalization. The experiments in this section use standard batch normalization and compare the resulting boost in model performance.

Figure 3.5 shows the training and validation accuracy of the VGG16 and ResNet34 networks with batch normalization layers. The results show that with batch normalization, both models with all activation functions are able to converge on both the training and validation datasets. However, there is a noticeable performance gap between the models.

Unlike the results in the previous section, the GeLU networks slightly outperform the ReLU networks, suggesting that GeLU has a better ability to approximate the objective function but is less robust compared to ReLU. Among the four models, the Sigmoid networks are shown to be unstable on the validation dataset during



(a) VGG16 Model



(b) ResNet34 Model

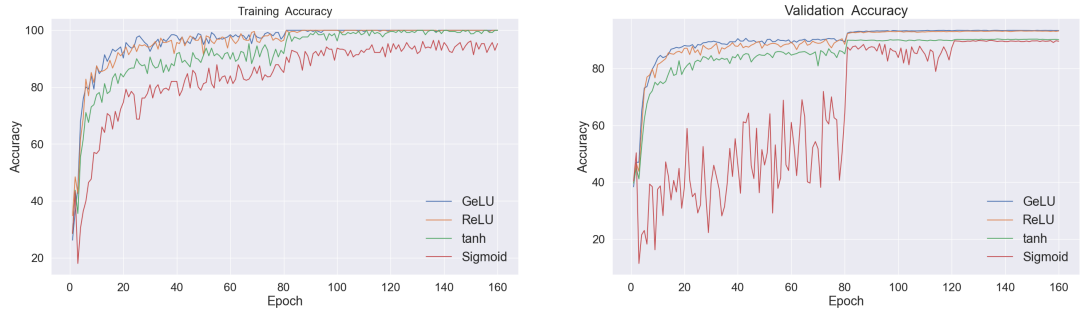
Figure 3.5: Trainning and Validation Accuracy of VGG16 network and ResNet34 Network with batch normalization but without weight initialization.

training. Specifically, while the training accuracy improves, the validation accuracy can sometimes drop rapidly. This implies that models with the Sigmoid activation function have worse generalization ability compared to others. This finding is further explored in Chapter 5.2 with the help of pattern similarity, introduced in Chapter 5.1.

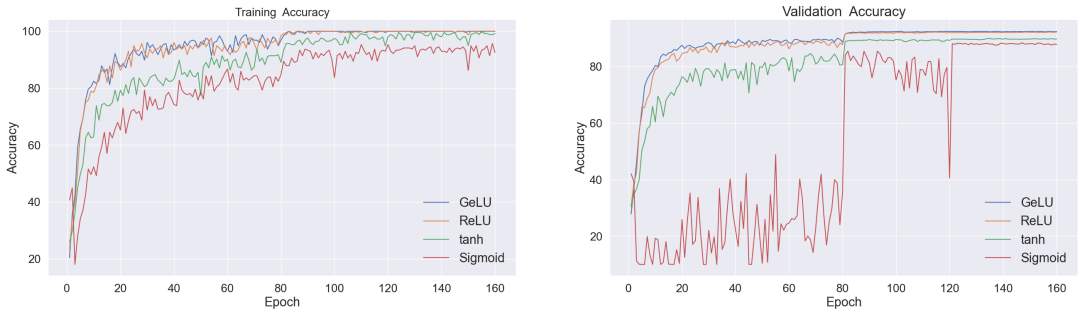
3.1.3.4 Benchmark + Init + BN

The last set of experiments applies both weight initialization and batch normalization to explore whether the performance gap between models with different activation functions still exists. Figure 3.6 presents the training and validation accuracy of the models.

Similar to Figure 3.5, all the models are able to converge by the end of training. The GeLU networks again show better validation accuracy than the ReLU networks,



(a) VGG16 Model



(b) ResNet34 Model

Figure 3.6: Trainning and Validation Accuracy of VGG16 network and ResNet34 Network with batch normalization and weight initialization.

as seen in Figure 3.5. Moreover, after introducing weight initialization, the validation accuracies of the Sigmoid networks, although still suffering from overfitting, perform better than in the previous section. This indicates that the Sigmoid networks are more heavily dependent on weight initialization.

The setup of this set of experiments, with batch normalization and weight initialization, is widely recognized as a default structure in recent literature. As expected, there is a noticeable performance gap between models with different activation functions, as indicated by the training and validation accuracy illustrated above.

3.1.4 Performance Summary

This section summarizes the experiments presented above concerning the effects of weight and gradient modification techniques. Based on the graphs of training and validation accuracy, the following statements can be concluded:

- For all activation functions, deep neural networks without any weight modification techniques fail to converge.
- ReLU networks are the most robust models with respect to the distribution of weights, as they are able to provide comparable accuracy on validation datasets with either weight initialization or batch normalization in both structures.
- GeLU networks can outperform ReLU networks, as discussed in [251], when the forward pre-activation and backward gradients are properly distributed. However, GeLU is less robust than ReLU when weights are poorly initialized.
- Tanh networks are also able to converge with only weight initialization, which implies that issues caused by saturation can be mitigated to some extent. Therefore, saturation is not the sole determinant of the poorer performance of these activations.
- Sigmoid networks have the worst performance in the above experiments. While the networks are able to converge with the modifications introduced, they exhibit a noticeable accuracy gap compared to other models. Additionally, the validation accuracy of Sigmoid networks oscillates rapidly as training accuracy increases, indicating that Sigmoid networks lack generalization regardless of the network architecture and experiment settings.

Table 3.2 and Table 3.3 present the validation accuracy for VGG16 and ResNet34 across all the experiment sets. The results for ResNet34 are slightly lower than those for VGG16, which is likely due to the down-sampling layers of ResNet34 in the early stages. Given the small size of the CIFAR10 dataset (32×32), a rapid decrease in

	Sigmoid	Tanh	ReLU	GeLU
Benchmark	10.00	10.00	10.00	10.00
Benchmark + Init	10.00	87.15	89.40	10.00
Benchmark + BN	82.87	85.81	86.76	86.83
Benchmark + BN + Init	89.62	90.16	93.22	93.39

Table 3.2: Validation Accuracy of VGG16.

	Sigmoid	Tanh	ReLU	GeLU
Benchmark	10.00	10.48	10.00	10.00
Benchmark + Init	10.00	67.41	87.42	86.42
Benchmark + BN	84.52	84.21	85.05	85.28
Benchmark + BN + Init	87.79	89.70	92.15	92.34

Table 3.3: Validation Accuracy of ResNet34.

sample size can make it challenging for the network to extract features. However, this does not affect the analysis in this section, which focuses on how deep network structures affect the performance of different activation functions.

For Sigmoid networks, the accuracy of VGG16 is 89.62% but drops by 1.83% on ResNet34. In contrast, the accuracy decline for Tanh, ReLU, and GeLU is 0.46%, 1.07%, and 1.05%, respectively. This indicates that Sigmoid networks still struggle to provide accurate estimations as the network depth increases. However, the drop in accuracy for Tanh networks is lower than that for ReLU and GeLU.

In this section, we verify the existing literatures by comparing the performance of several benchmark models with different activation functions. It is found that there exists the ReLU and GeLU function outperforms Sigmoid and Tanh function with and without the use of batch normalization and weights initialization.

3.2 Re-Investigate Over-Saturation

In the previous section, it was confirmed that a distinguishable performance gap exists between neural networks with different activation functions, as expected. Notably, this gap also varies depending on the model structure. The first step in exploring this observation is to determine whether the *vanishing gradient issue* still persists in the modified models.

The vanishing gradient issue with Sigmoid activation refers to the phenomenon where, when the inputs of neurons are extremely large or small, the Sigmoid function saturates in regions where gradients are nearly zero. This leads to a failure in the backpropagation process, causing the network to fail to converge. Since its identification, this issue has been widely accepted as the cause of poor performance in Sigmoid and other activation functions, such as Tanh. However, this issue was observed in the early days, before the introduction of other deep neural network techniques, such as batch normalization [252] and weight initialization techniques [17], [18]. The empirical results from Table 3.2 and Table 3.3 suggest that with these modifications, neural networks with saturated activations can converge, but still exhibit a performance gap compared to other activation functions. Therefore, it is necessary to investigate whether the problem still exists.

With this objective, this section re-examines the behavior of over-saturated activation functions in different model structures. In particular, it monitors and compares the weights and gradients of models with different structures during training. As discussed in previous works, the problem persists if the gradients of the network tend to be small for saturated functions, but it can be disregarded if the models exhibit similar behavior in terms of weights and gradients.

This study begins with simple fully connected models trained on the MNIST dataset using Sigmoid and ReLU activation functions. It is shown that the post-activation values, weights, and gradients are distributed around the activated regions for both ReLU and Sigmoid networks. This suggests that the vanishing gradient issue for this simple network is properly addressed. However, it is observed that the

weight updates in the Sigmoid network are less stable than in the ReLU network during training.

To extend the results to deeper networks, this section then explores the pre-activation values, weights, gradients, and the weight-to-gradient ratio for VGG16 and ResNet34 models. Similar to the DNN network, it is found that the vanishing gradient issue is properly addressed, but the stability of training the network is more challenging for neural networks with deeper structures.

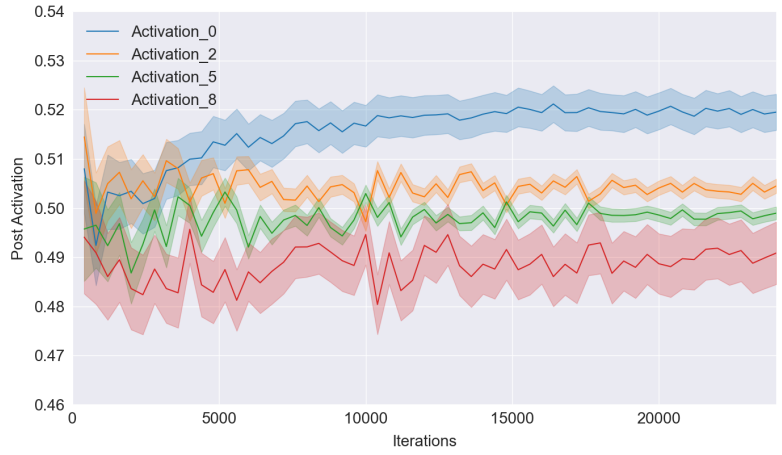
3.2.1 Fully Connected Network

3.2.1.1 Experiment Settings.

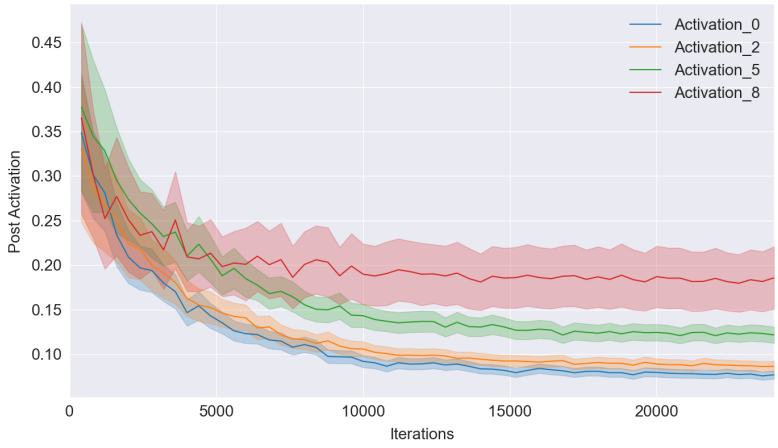
A stacked fully connected neural network (FCNN) is used to examine the over-saturation issue, in line with the experiments that initially discovered the problem. Both the Sigmoid and ReLU networks consist of 9 hidden layers, each with 256 neurons, and a softmax logistic regression for the output layer. The cost function is cross-entropy loss, which is widely adopted in classification tasks. Both networks are trained on the MNIST dataset for 24,000 iterations with a batch size of 128, using the SGD optimizer and a linear rate scheduler decaying from 0.1 to 0.0001.

3.2.2 Post Activation

Figure 3.7 compares the post-activation values of the Sigmoid and ReLU networks trained on the MNIST dataset. The solid lines and shaded areas represent the mean and variance of the post-activation values for 128 fixed test samples and 64 fixed neurons for each layer. It can be observed that the vanishing gradient issue no longer exists with the recently proposed techniques. For the Sigmoid network, the post-activation values across all layers have a mean around 0.5 and a variance around 0.05, with layers 8 and 0 showing the highest variance, while the others exhibit relatively lower variance. For the ReLU network, the mean post-activation values are around 0.1, and the variance decreases as the model converges. Notably,



(a) Post Activatoin of Sigmoid net



(b) Post Activatoin of ReLU net

Figure 3.7: The mean and variance of post-activation values of a network with 9 layers, with 256 neurons within each layer.

these results are consistent across different neuron sets and test sample sets.

These observations indicate that when batch normalization is introduced to the model, the post-activation values and gradients of the Sigmoid network are no longer saturated. This suggests that the vanishing gradient issue in the Sigmoid network can be mitigated by re-centering and re-scaling the values in each layer. However, despite this improvement, the performance of the Sigmoid network remains far

behind that of state-of-the-art activation functions. This suggests that there are other issues preventing Sigmoid activation from achieving optimal performance.

3.2.2.1 Weights and Gradients.

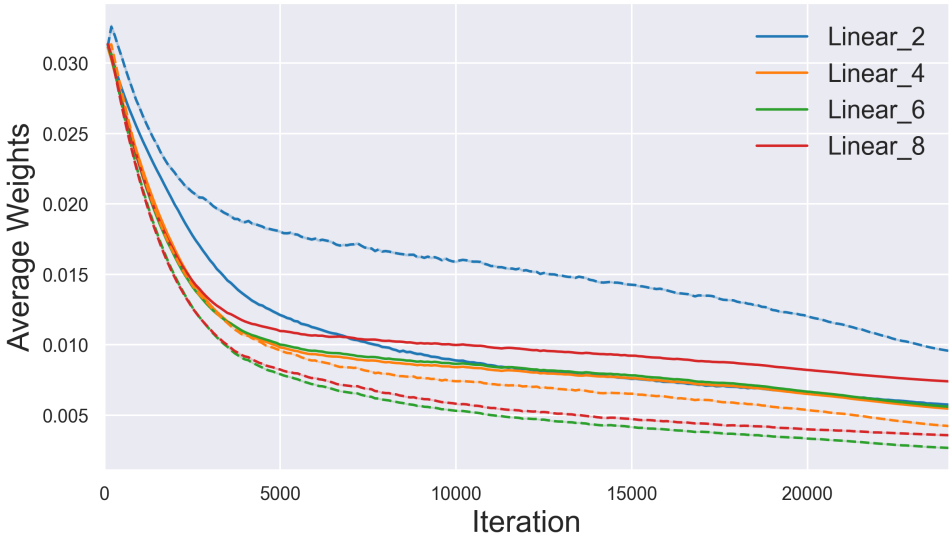


Figure 3.8: Unit wise average gradient of fully connected network where the solid lines are Sigmoid network and dashed lines are ReLU network

Your text is clear and well-structured. Here's a slightly refined version for clarity and flow:

Figure 3.8 and Figure 3.9 present the absolute values of unit-wise gradients and weights. The experiment settings are the same as described in Section 3.2.1.1. Every 200 steps, the weights and gradients of each layer are recorded and averaged. The solid lines and dashed lines show the ratios for the Sigmoid and ReLU networks, with different layers represented by lines in various colors.

Generally, during the early and middle stages of training, the gradients and weights of both networks exhibit similar behavior. Figure 3.8 presents the weights of the ReLU and Sigmoid networks. The averaged weights in the Sigmoid network remain consistent across different layers during training, while in the ReLU network, the weights are slightly higher in layer 2. Figure 3.9 shows the unit-wise average

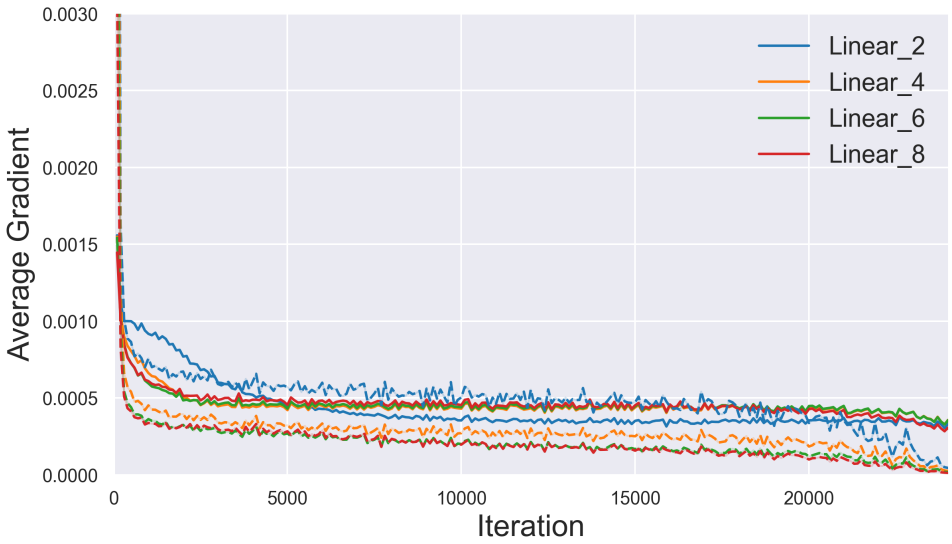


Figure 3.9: Unit wise average gradient of fully connected network where the solid lines are Sigmoid network and dashed lines are ReLU network.

gradients of the networks. The only notable difference is that the gradients in the ReLU network are slightly lower than those in the Sigmoid network in the deeper layers during training. As the networks converge, the gradients across different layers do not show any distinguishable differences. This suggests that with batch normalization and weight initialization, the gradients can be properly propagated back to the shallow layers.

Combining the results from Figure 3.7, Figure 3.8, and Figure 3.9, it can be concluded that the neurons in both networks are activated, implying that the vanishing gradient issue is no longer the primary cause of the performance gap between the networks.

One notable finding from Figure 3.9 is that the averaged absolute value of the gradients in the Sigmoid network is larger than in the ReLU network during the late stage of training. However, by this stage, both the training and validation accuracy show little improvement. This raises the question: Are these weight updates truly effective in training Sigmoid networks?

3.2.2.2 Gradients to Weights Ratio

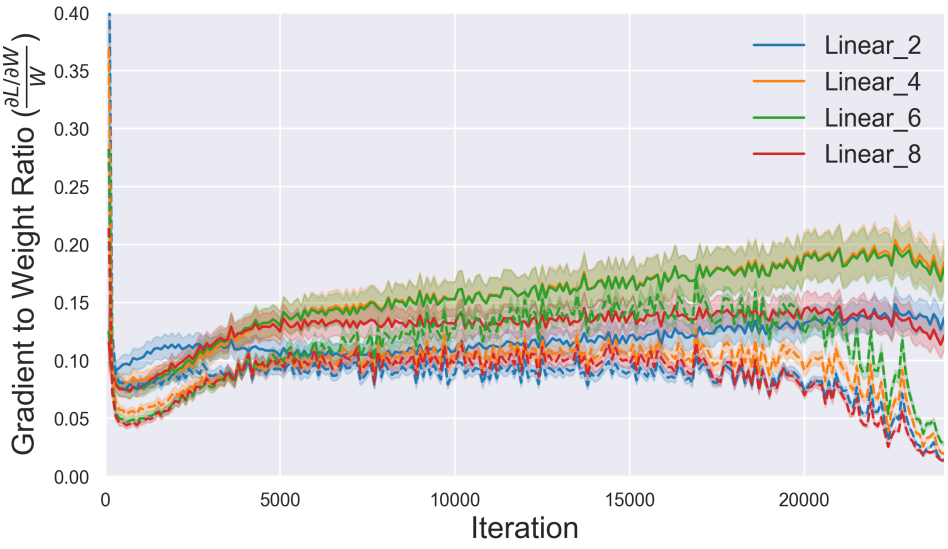


Figure 3.10: Gradient to weight ratio ($\frac{\partial L/\partial W}{W}$) of fully connected network where the solid lines are Sigmoid network and dashed lines are ReLU network..

To study the stability of network training, the gradient-to-weight ratio is further analyzed for the fully connected neural network.

Figure 3.10 compares the $|\frac{\partial L/\partial W}{W}|$ ratio. At each step, after backpropagation, we compute $|\frac{\partial L/\partial W}{W}|$ for each neuron and average the value every 200 steps. Similar to previous sections, the lines represent the mean, and the shaded areas represent the variance of the ratio of neurons at each layer. The solid lines and dashed lines show the ratio for the Sigmoid and ReLU networks. Due to the high variance in the ratio for the Sigmoid network, we scale the variance to 0.25 for both networks for aesthetic reasons.

The variance of the gradient-to-weight ratio can be viewed as an indicator of training stability. For the ReLU network, the lower variance of the ratio suggests that the gradients are proportional to their current weights at the neuron level, making the current weight values a scale factor for the learning rate. Notably, the variance gradually decreases as training progresses. However, in the Sigmoid network, the gradient-to-weight ratio of neurons diverges significantly throughout

training. This observation suggests that during training, the gradients in the Sigmoid network are less correlated with their weights and are less stable compared to the ReLU network.

Since the weights and gradients for both models perform similarly during training, the differences in the unit-wise variance of the gradient-to-weight ratio between the two networks are not caused by the vanishing gradient issue but by the nature of the activation function.

To further understand this, Theorem 1 in Section 3.3 provides a theoretical explanation, showing that the training stability of the Sigmoid network is inherently less stable. As the model depth increases, the weight update ΔW_i is more likely to be influenced by its current weight W_i rather than the training samples, leading to reduced stability during training.

3.2.3 Deep Networks

Before delving into the theoretical analysis, it is essential to verify whether similar observations can be made for neural networks with deeper structures. This section investigates the same metrics as those in Section 3.2.1, but applied to VGG16 and ResNet34 models with additional activation functions. Each model is trained on the MNIST dataset in the same manner as in Section 3.1, with batch normalization and weight initialization introduced. The objectives of this section are:

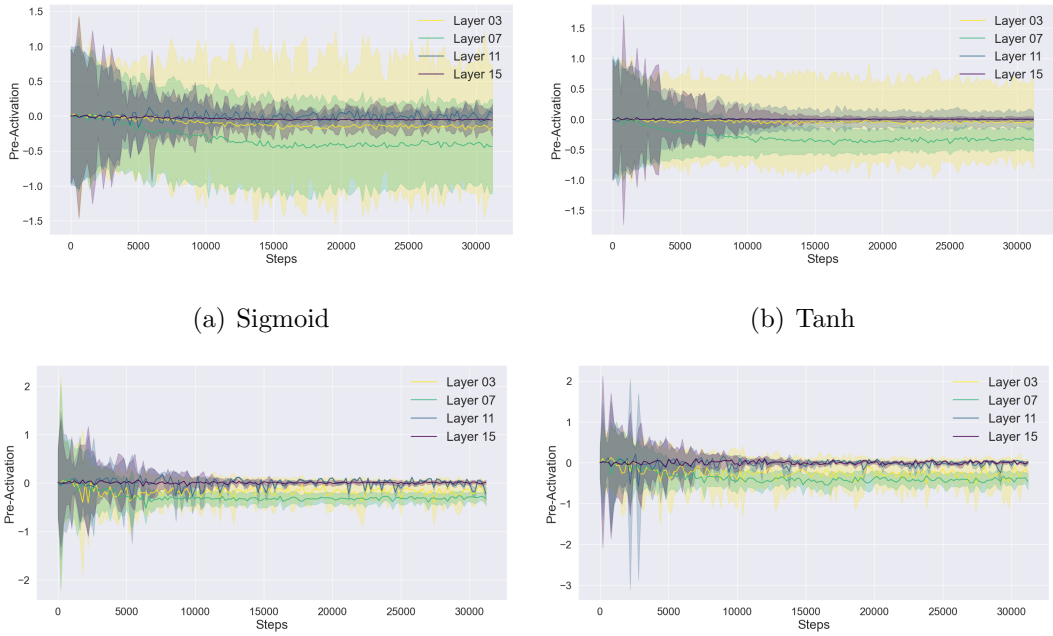
- Examining the vanishing gradient issue in neural networks with deeper structures to generalize the conclusions.
- Comparing the metrics for neural networks with different activation functions to provide insights into how activation functions affect training stability.

3.2.3.1 Experiment Settings

The experiments in this section monitor the learning dynamics of VGG16 and ResNet34 on the CIFAR10 dataset. Each model is trained for 160 epochs using

the SGD optimizer.

3.2.3.2 Pre-Activation



(a) Sigmoid

(b) Tanh

(c) ReLU

(d) GeLU

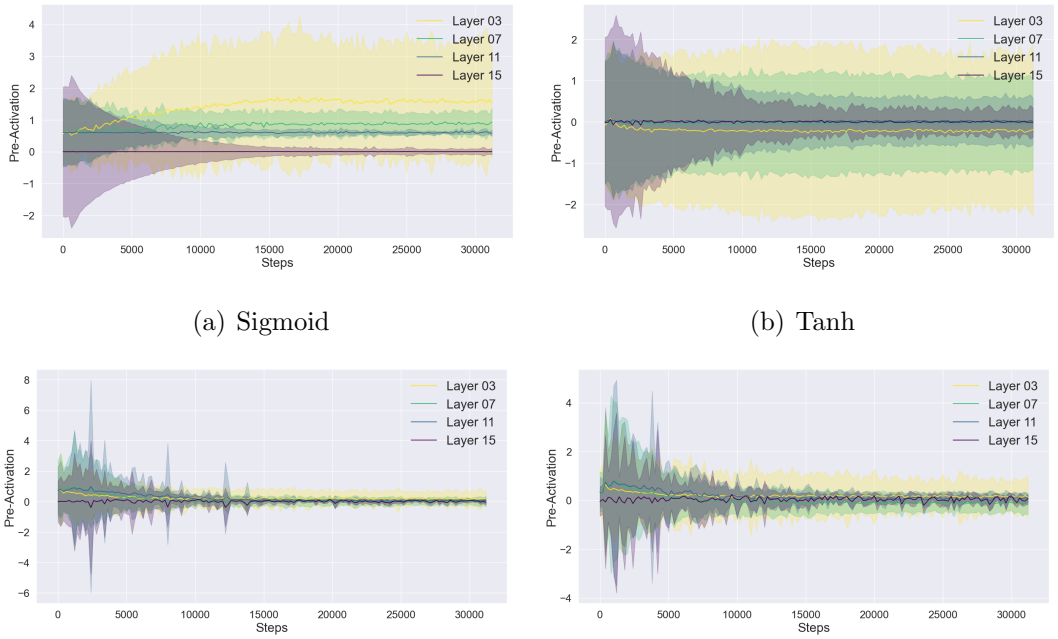
Figure 3.11: Pre Activation Value of VGG16

Figure 3.11 shows the pre-activation values for the VGG16 network at different layers. The solid lines and shaded areas represent the mean and variance of the pre-activation values of the network. From light to dark, different colors represent the values for layer 3, layer 7, layer 11, and layer 15.

The most notable observation is that, with the help of batch normalization, the averaged pre-activation values for all the networks are centered around 0 across all layers. For all models, the mean for layer 7 is slightly below 0 but hovers around -0.2. These results are similar to those observed in the fully connected network.

On the other hand, the variance of pre-activation values for saturated activation functions (Sigmoid and Tanh) is larger than that of non-saturated activation functions (ReLU and GeLU), especially in the shallow layers (layer 3 and layer 7). However, the variance for the Sigmoid and Tanh networks is still less than one,

which means that the majority of neurons are not saturated, despite having a larger variance than their non-saturated counterparts. Additionally, it is found that for the deeper layers, the variance gradually converges in most of the networks.



(a) Sigmoid

(b) Tanh

(c) ReLU

(d) GeLU

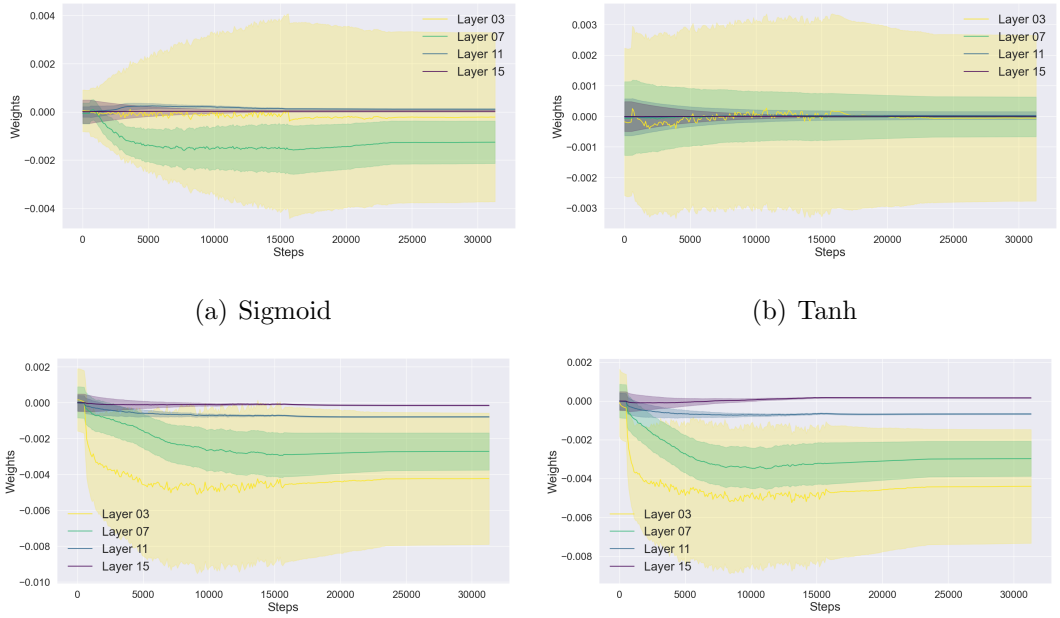
Figure 3.12: Pre Activation Value of ResNet34

Figure 3.12 shows the pre-activation values for ResNet34 at several layers. Most of the observations from the VGG16 model can also be seen in ResNet34. However, the variance in ResNet34 is larger than in VGG16.

When combining the variance with model performance, it can be observed that the variance in the Sigmoid and Tanh networks is larger than in the VGG16 network, but the gap in validation accuracy between these networks and the ReLU and GeLU networks is relatively the same. Moreover, the variance in the GeLU network is larger than in the ReLU network for both VGG16 and ResNet34, yet GeLU slightly outperforms ReLU. This suggests that the variance of pre-activation values is not a determining factor in model performance.

3.2.3.3 Weights and Gradients

Similar to the approach in Section 3.2.1, the next metrics to monitor are the weights and gradients of the network during training.



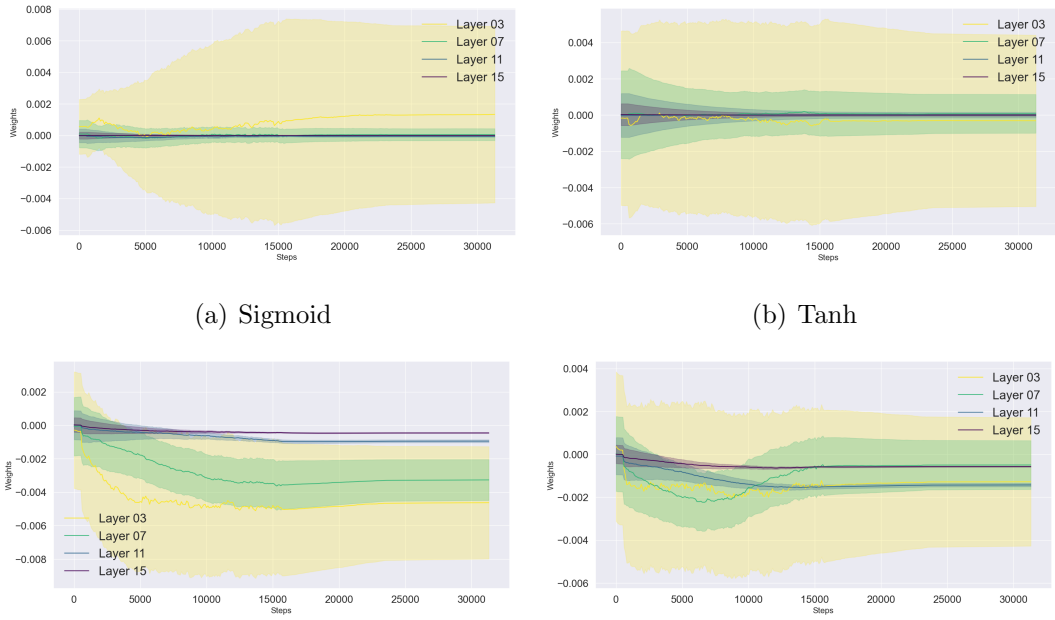
(a) Sigmoid (b) Tanh

(c) ReLU (d) GeLU

Figure 3.13: Weights of VGG16

Figures 3.13 and 3.14 compare the mean and variance of weights in VGG16 and ResNet34 models. For all activation functions, both VGG16 and ResNet34 exhibit relatively low-magnitude weights with low variance in the deeper layers (layer 11 and layer 15). Additionally, as the network converges, the variance of weights in deep layers becomes even smaller, suggesting that the predictions made by the neural network are precise and can be easily affected by perturbations.

Conversely, for both network structures, the weights in shallow layers show significantly larger variance regardless of the activation function. This occurs because shallow layers serve as feature extractors that process the input to the model. Compared to the input for deep layers, the input values range from (0, 1) and have a larger magnitude than the pre-activation values in deep layers, as shown in Figures 3.11 and 3.12. Therefore, the high variance of weights in shallow



(a) Sigmoid

(b) Tanh

(c) ReLU

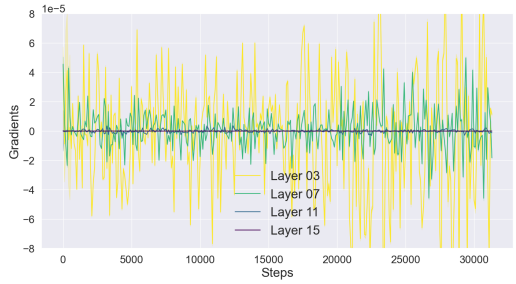
(d) GeLU

Figure 3.14: Weights of ResNet34

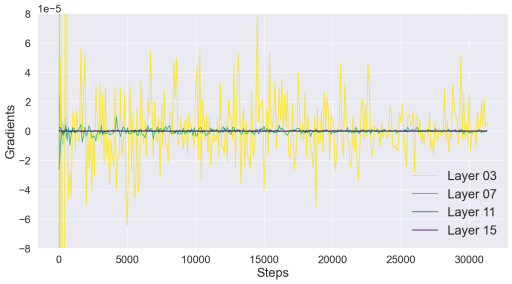
layers is common across all models and cannot be considered a factor contributing to performance differences.

In general, the model weights of neural networks during training do not show a clear pattern that can be linked to model performance, making it difficult to draw insights into the impact of neural network activations. However, the graphs of gradients are more noteworthy than those in Section 3.2.1.

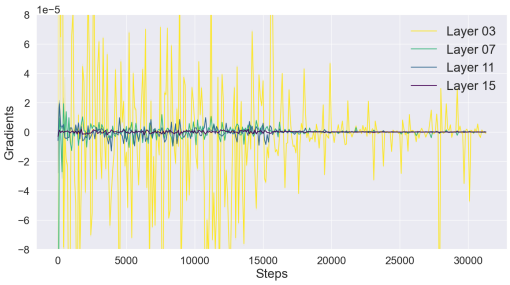
Figure 3.15 presents the gradients of VGG16 models with different activation functions during training. For the deep layers, models with different activation functions exhibit low gradients throughout the training. The gradients in the shallow layers oscillate rapidly during the early stages of training, indicating that the models are actively searching for optimal parameters. The magnitude of gradients in the ReLU and GeLU networks gradually decreases as the models converge. However, in models with saturated activation functions, the gradients remain highly volatile even in the later stages of training. Compared to the Tanh network, the gradients of the Sigmoid network are even more unstable during late-stage training.



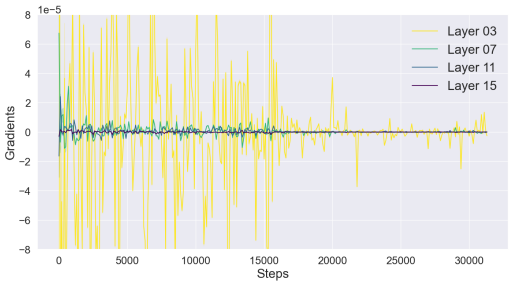
(a) Sigmoid



(b) Tanh

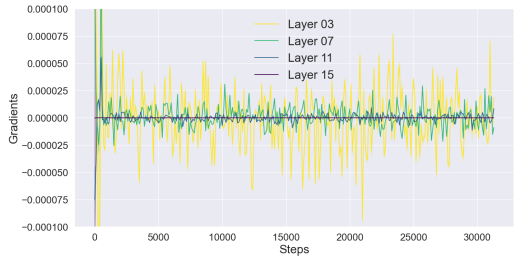


(c) ReLU

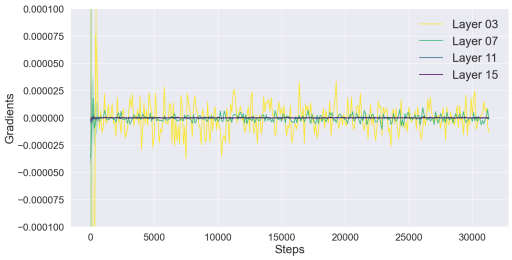


(d) GeLU

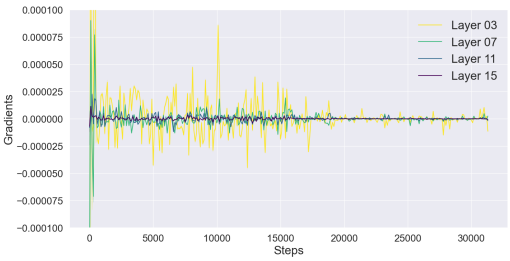
Figure 3.15: Gradients of VGG16



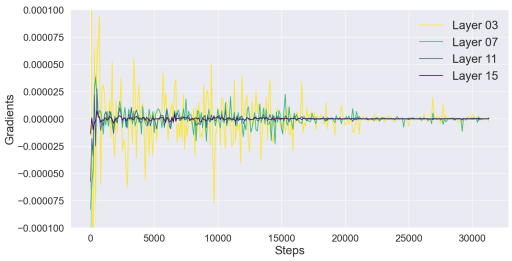
(a) Sigmoid



(b) Tanh



(c) ReLU



(d) GeLU

Figure 3.16: Gradients of VGG16

A similar pattern can be observed in ResNet34, as shown in Figure 3.16. The gradients in the deep layers remain low for all models during training, while the gradients in the shallow layers behave differently. As the models converge, the ReLU and GeLU networks gradually stabilize with low average gradients, while the Tanh network maintains high gradients throughout the entire training process. The gradients in the Sigmoid network are the most unstable and even increase during training, as observed in the VGG16 network.

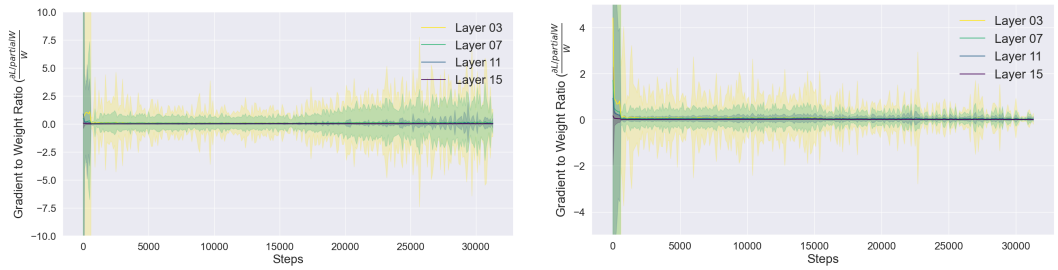
The dynamic behavior of weights and gradients during training differs between saturated activation functions and others. Intuitively, as the model converges, the gradient changes should be minimal if the model is approaching optimal parameters. However, in both the Sigmoid and ReLU networks, the gradients oscillate rapidly throughout training and even become higher in the later stages. This suggests that training saturated models is unstable and fails to find optimal parameters.

3.2.3.4 Ratios

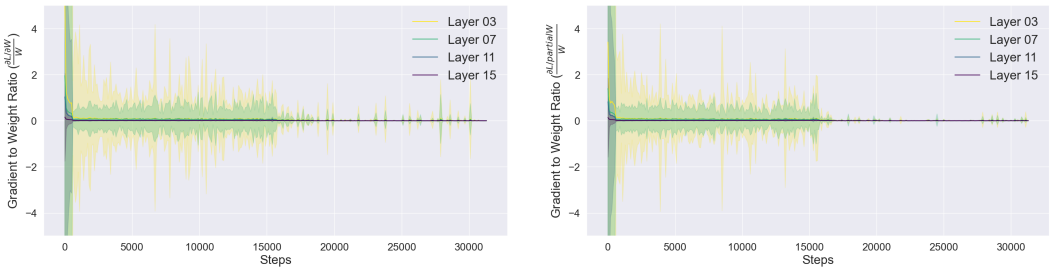
This set of experiments compares the ratio between weights and gradients to explore whether the gradients are affected by the weights. The gradients and weights are recorded every 200 steps during training. The gradient-to-weight ratio is computed for each parameter. The solid lines represent the mean gradient-to-weight ratio, while the shaded areas represent the variance for the model.

Figure 3.17 presents the gradient-to-weight ratio of VGG16 with different activation functions. It can be observed that for both ReLU and GeLU, the variance of the ratio is close to 0, with minor outliers in the late stages of training. This implies that the gradient is largely independent of the model weights. For the Tanh network, the gradient-to-weight ratio has a variance of around 1.5 to 2 in the early stages of training, gradually decreasing to 0.5. The Sigmoid network shows the most instability, with the variance of the ratio increasing to around 3.

A similar pattern can be observed for ResNet34, as shown in Figure 3.18. Compared to VGG16, both the ReLU and GeLU networks exhibit lower variance

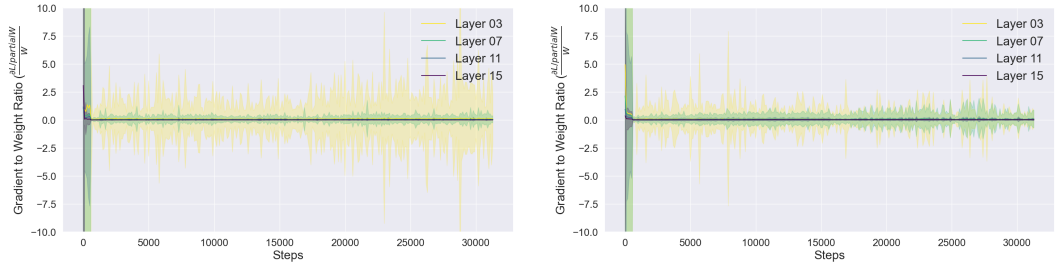


(a) Sigmoid (b) Tanh

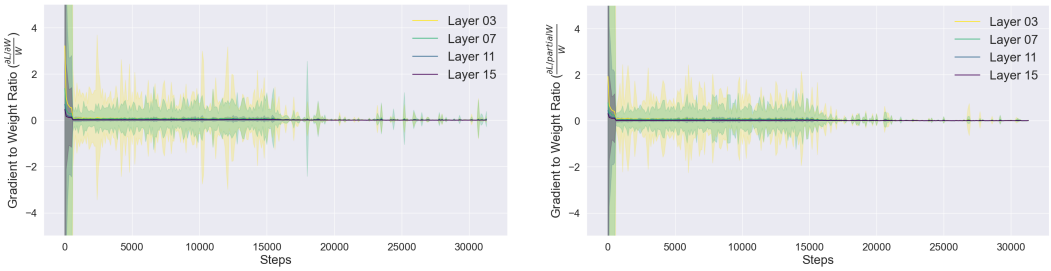


(c) ReLU (d) GeLU

Figure 3.17: Gradients to Weights Ratio of VGG16



(a) Sigmoid (b) Tanh



(c) ReLU (d) GeLU

Figure 3.18: Gradients to Weights Ratio of ResNet34

during the early stages of training on ResNet34. The Tanh network shows higher variance in the middle layers (Layer 7), while the Sigmoid network has slightly lower variance. In general, the variance for saturated activation functions remains high in ResNet34.

The plots in this section suggest that the training of models with saturated activation functions is less stable than that of models with other activation functions. This instability indicates that the models may struggle to find optimal parameters during training. To further understand these observations, the next section presents a theoretical analysis of the learning dynamics of feedforward neural networks.

3.3 Learning Dynamic of Neuron Networks

Theoretically, a deep neural network with non-linear activation functions is capable of approximating any function. However, the comparison of model performance in the previous section shows that the Sigmoid and Tanh networks fail to converge in deeper structures. To verify whether this issue is still caused by the *vanishing gradient problem*, the distribution of pre-activation values, weights, and gradients of all networks during training is further illustrated.

It is found that the pre-activation values of the Sigmoid and Tanh networks are located within the active region. Moreover, the gradients of saturated activation functions no longer vanish in shallow layers, indicating that batch normalization and weight initialization can effectively solve the vanishing gradient issue. However, it is also observed that the gradients in the Sigmoid and Tanh networks are less stable than in the ReLU and GeLU networks. In particular, the gradient-to-weight ratio in these networks exhibits larger variance throughout training, even after the learning rate decreases. This implies that the weights in these networks oscillate during training.

To understand the cause of this observation and its impact on model convergence, this section investigates the training dynamics of neural networks with respect to

activation functions through a theoretical analysis.

The main result of this section is presented in Theorem 1, which describes the relationship between gradients and weights in neural networks during training. The result is derived through several steps:

- First, it begins with the forward propagation of the neural network, focusing on introducing the notations used in the theoretical analysis in this section.
- Second, it illustrates the dependencies of gradients on the parameters by computing the partial derivative of $\frac{\Delta W^{(i)}}{W^{(i)}}$. Specifically, the gradient of the parameter $W^{(i)}$ is expressed in terms of the parameter $W^{(i)}$ itself. By grouping the variables, it shows how the training process is affected by the current parameters.
- Finally, it establishes a connection between the variables discussed in the previous section and the choice of activation function. This explains the oscillation of gradients and the gradient-to-weight ratio in the Sigmoid and Tanh networks observed in Section 3.1.

3.3.1 The Forward Propagation

Let $\mathcal{N}$ be a feedforward neural network with activation function π defined on the input space $\mathbb{R}^n$. The mapping function can be expressed as a composition of functions: $f = h_d \circ h_{d-1} \circ \cdots \circ h_1$, where h_i is a layer consisting of a linear transformation ϕ_i and a non-linear activation function π_i.

Given a linear transformation ϕ_i, such as a fully connected layer, convolutional layer, or batch normalization layer, it can be viewed as applying a matrix multiplication followed by a shift. Therefore, ϕ_i can be represented by a weight matrix $W^{(i)}$ and a bias $b^{(i)}$.

The training of the neural network aims to minimize the prediction loss on the training dataset. A training sample is denoted as $(\mathbf{x}, y)$, where $\mathbf{x} \in \mathbb{R}^n$ is the observation and $y \in \mathbb{R}^C$ is the label of the sample. The neural network computes an

output $\hat{y} = f(\mathbf{x}) \in \mathbb{R}^c$, where $\mathbb{R}^c$ is the output space with a dimension of c. Since $\mathcal{N}$ is a stacked feedforward neural network, $f(\mathbf{x})$ is a composition of the functions at each block. Denote $x^{(i)}$, $y^{(i)}$, and $z^{(i)}$ as the input, output, and pre-activation of block i. Therefore, it can be expressed as:

$$\begin{aligned} z^{(i)}(\mathbf{x}) &= \phi_i(y^{(i)}(\mathbf{x})) = W^{(i)}x^{(i)}(\mathbf{x})^T + b^{(i)}(\mathbf{x}), \\ y^{(i)}(\mathbf{x}) &= \pi(z^{(i)}(\mathbf{x})), \\ x^{(i+1)}(\mathbf{x}) &= y^{(i)}(\mathbf{x}). \end{aligned}$$

The equations above illustrate the relationship between the inputs, outputs, and pre-activation of layer i:

- The linear transformation maps the input vector $x^{(i)}$ to the pre-activation vector $z^{(i)}$.
- The non-linear activation function π is then applied to the pre-activation $z^{(i)}$ to obtain the output vector $y^{(i)}$.
- The output of layer i serves as the input for layer $i + 1$.

Similarly, the input to the first layer of network $\mathcal{N}$ is the input to the network, and the output of the last layer of network $\mathcal{N}$ is the output of the network:

$$\begin{aligned} \mathbf{x}^{(1)} &= \mathbf{x}, \\ y^{(d)} &= f(\mathbf{x}), \end{aligned}$$

where $f(\mathbf{x})$ is the prediction of the neural network, the label of $\mathbf{x}$ is assigned as $\hat{y} = \arg\max_{i \in y} f_y(\mathbf{x})$. For a loss function L, the backward propagation process computes the gradients of the parameters with respect to the prediction loss $L(\hat{y}, y)$ and updates the parameters to minimize the loss accordingly.

To study the stability of the neural network during training, the following section considers the relationship between the gradient and the weight for a training sample $(\mathbf{x}, y)$. In particular, the next step discusses how the form of activation functions affects the update of parameters.

3.3.2 Stability of Backward Propagation

Given a loss function L, at each training step, the network first computes the loss $L(\hat{y}, y)$, where $(\mathbf{x}, y)$ is the training sample. The backward propagation process then aims to minimize the loss function by computing the optimal direction for adjusting the parameters to reduce the loss.

Considering the weights of layer i, the gradient of the weights in layer i is the partial derivative of $L(\hat{y}, y)$ with respect to the weights:

$$\frac{\partial L(\hat{y}, y)}{\partial W^{(i)}} = \frac{\partial L(\hat{y}, y)}{\partial y^{(i)}} \frac{\partial y^{(i)}}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial W^{(i)}}.$$

The change of weight $\Delta W^{(i-1)}$ regarding learning rate η is:

$$\Delta W^{(i)} = \frac{\partial L(\hat{y}, y)}{\partial y^{(i)}} \frac{\partial y^{(i)}}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial W^{(i)}} \eta = \frac{\partial L(\hat{y}, y)}{\partial y^{(i)}} \frac{\partial y^{(i)}}{\partial z^{(i)}} y^{(i-1)} \eta.$$

Consider the derivative of $\Delta W^{(i)}$. Since $\mathcal{N}$ is a feedforward network, the output of layer $i-1$ is independent of the weights of layer i. This implies that $\Delta W^{(i)}$ is a function of $\frac{\partial L(\hat{y}, y)}{\partial y^{(i)}}$ and $\frac{\partial y^{(i)}}{\partial z^{(i)}}$. The derivative of $\Delta W^{(i)}$ with respect to $W^{(i)}$ is:

$$\frac{\partial \Delta W^{(i)}}{\partial W^{(i)}} = C_1 \left[\frac{\partial^2 L}{\partial y^{(i)} \partial W^{(i)}} \frac{\partial y^{(i)}}{\partial z^{(i)}} + \frac{\partial L}{\partial y^i} \frac{\partial^2 y^{(i)}}{\partial z^{(i)} \partial W^{(i)}} \right], \tag{3.4}$$

where $C_1 = y^{(i-1)} \eta$. Since,

$$\begin{aligned} \frac{\partial L}{\partial y^{(i)}} &= \frac{\partial L}{\partial y^{(d)}} \prod_{j=i}^{n-1} \frac{\partial y^{(j)}}{\partial z^{(j)}} \frac{\partial z^{(j)}}{\partial x^{(j)}} \\ &= \frac{\partial L}{\partial y^{(d)}} \prod_{j=i}^{d-1} \frac{\partial y^{(j)}}{\partial z^{(j)}} W^{(j)}. \end{aligned} \tag{3.5}$$

At each training step, assume that the weights of model is fixed parameter. Given training sample $(\mathbf{x}, y)$, Equation 3.5 can be written as:

$$\frac{\partial L}{\partial y^{(i)}} = C_2 \frac{\partial L}{\partial y^n}$$

where C_2 is defined as the value of $C_2(x, y, \theta) = \prod_{j=i}^{d-1} \frac{\partial y^{(j)}}{\partial z^{(j)}} W^{(j)}$, which is a function of training sample $\mathbf{x}, y$ and parameters $W^{(j)}, j = i+1, \ldots, d$. As $\frac{\partial y^{(j)}}{\partial z^{(j)}}$ and $W^{(j)}$ is

irrelevant with $W^{(i)}$, the former part of Equation 3.4 can be arranged as:

$$\frac{\partial^2 L}{\partial y^{(i)} \partial W^{(i)}} \frac{\partial y^{(i)}}{\partial z^{(i)}} = \frac{\partial \frac{\partial L}{\partial y^{(i)}}}{\partial W^{(i)}} = \frac{\partial^2 L}{\partial y^{(d)} W^{(i)}} \prod_{j=i}^{n-1} \frac{\partial y^{(j)}}{\partial z^{(j)}} W^{(j)}.$$

The latter part of Equation 3.4 can be written as:

$$\frac{\partial L}{\partial y^i} \frac{\partial^2 y^{(i)}}{\partial z^{(i)} \partial W^{(i)}} = \frac{\partial L}{\partial y^{(d)}} \frac{\partial^2 y^{(i)}}{\partial z^{(i)} \partial W^{(i)}} \prod_{j=i}^{d-1} \frac{\partial y^{(j)}}{\partial z^{(j)}} W^{(j)}$$

Then Equation 3.4 then can be formatted as:

$$\frac{\partial \Delta W^{(i-1)}}{\partial W^{(i-1)}} = C_1 C_2 \left[\frac{\partial^2 L}{\partial y^n \partial W^i} \frac{\partial y^i}{\partial x^{(i)}} + \frac{\partial L}{\partial y^n} \frac{\partial^2 y_i}{\partial x^{(i)} \partial W^{(i-1)}} \right] \tag{3.6}$$

Now we consider the loss function L. For the activations function with linear derivatives, such as widely adopted mean square error (MSE) and softmax cross-entropy loss, their derivatives can be written as

$$\frac{\partial L}{\partial y^{(d)}} = a \times y^{(d)} + b, \tag{3.7}$$

where $\hat{y}$ and y are predicted vector and ground truth vector, a and b are coefficients of the derivative of the loss function.

3.3.2.1 Activation and Stability

As the last part of the discussion, this section aims to explain the stability of the training of neural network and connects it with the activation function.

First, consider the former part of Equation 3.6 given a piece wise linear activation. Since F is a stacked model, W^i is multiple once during the forward propagation, so $y^n \propto W^{(i-1)}$. By combining Equation 3.7 we know that the first multiplier of the former part is irrelevant with $W^{(i-1)}$. On the other hand, since $\frac{\partial y^i}{\partial x^{(i)}}$ is a constant, the latter part of Equation 3.6 is zero. Recall that the variable C_1 and C_2 are also irrelevant with $W^{(i-1)}$. For a network with piece-wise linear activation, Equation 3.6 can be written as:

$$\frac{\partial \Delta W^{(i-1)}}{\partial W^{(i-1)}} = C_0(x, y, \theta \backslash \{W^{(i-1)}\})$$

where

$$C_0 = C_1 C_2 \frac{\partial^2 L}{\partial y^n \partial W^i} \frac{\partial y^i}{\partial x^{(i)}}$$

is a function of x, y and $\theta \backslash \{W^{(i-1)}\}$. Based on above analysis, we now can yield our theorem for comparing the learning dynamic of two activations.

Theorem 1. *Given a deep neural network with piece-wise linear activation, if the derivative of loss function $L(\hat{y}, y)$ is linear, then for a given training sample (x, y), $\Delta W^i \propto W^i$, $i = 1, 2, ...n$, where W^i is the weight of ith layer.*

Proof. *Given a d layer deep neural network $\mathcal{N}$ with ReLU activation and training sample $(\mathbf{x}, y)$, denote $x^{(k)}$, $y^{(k)}$ and $\theta^k = \{W^k, b^k\}$ as input, output, and parameters of layer k, $k = 1, \ldots, N$. Note that $y^0 = x$ and $y^N = \hat{y}$. For the forward pass we have:*

$$x^k = W^{(k-1)} y^{(k-1)} + b^{k-1},$$

$$y^{(k)} = f^k(x^k),$$

where weight $W^{(i-1)} \in R^{n_i \times n_{i-1}}$ and the bias $b^{i-1} \in R^{n_l}$. Given a loss function $L(\hat{y}, y)$ and a training instance (x_1, y_1), the change of weight $\Delta W^{(i-1)}$ regarding learning rate η is:

$$\begin{aligned} \Delta W^{(i-1)} &= \frac{\partial L}{\partial W^{(i-1)}} \eta = \frac{\partial L}{\partial y^i} \frac{\partial y^i}{\partial x^{(i-1)}} \frac{\partial x^{(i-1)}}{\partial W^{(i-1)}} \eta \\ &= \frac{\partial L}{\partial y^i} \frac{\partial y^i}{\partial x^{(i-1)}} y^{(i-1)} \eta. \end{aligned}$$

where $\Delta W^{(i-1)}$ is a function of x^i, y^i, θ. Now we consider the derivative of $\Delta W^{(i-1)}$:

$$\frac{\partial \Delta W^{(i-1)}}{\partial W^{(i-1)}} = C_1 \left[\frac{\partial^2 L}{\partial y^i \partial W^{(i-1)}} \frac{\partial y_i}{\partial x^{(i-1)}} + \frac{\partial L}{\partial y^i} \frac{\partial^2 y_i}{\partial x^{(i-1)} \partial W^{(i-1)}} \right]$$

where $C_1 = y^{(i-1)} \eta$. Since

$$\frac{\partial L}{\partial y^{(i)}} = \frac{\partial L}{\partial y^{(i+1)}} \frac{\partial y^{(i+1)}}{\partial x^{(i)}} \frac{\partial x^{(i)}}{\partial y^{(i)}} = \frac{\partial L}{\partial y^{(i+1)}} \frac{\partial y^{(i+1)}}{\partial x^{(i)}} W^{(i)}$$

we have

$$\frac{\partial L}{\partial y^{(i)}} = \frac{\partial L}{\partial y^n} \prod_{j=i}^{n-1} \frac{\partial y^{(j+1)}}{\partial x^{(j)}} W^{(j)} \tag{3.8}$$

then for a fixed model F_θ and a training sample (x, y),

$$\frac{\partial L}{\partial y^{(i)}} = C_2 \frac{\partial L}{\partial y^n}$$

where C_2 is a constant defined, as the value of $C(x) = \prod_{j=i}^{n-1} \frac{\partial y^{(j+1)}}{\partial x^{(j)}} W^{(j)}$ with an input x. Therefore, equation 3.4 can be replaced as:

$$\frac{\partial \Delta W^{(i-1)}}{\partial W^{(i-1)}} = C_1 C_2 \left[\frac{\partial^2 L}{\partial y^n \partial W^i} \frac{\partial y^i}{\partial x^{(i-1)}} + \frac{\partial L}{\partial y^i} \frac{\partial^2 y_i}{\partial x^{(i-1)} \partial W^{(i-1)}} \right]$$

As the ReLU activation function is designed as:

$$relu(x) = \begin{cases} x, & x > 0 \\ 0, & elsewise \end{cases}$$

the derivative of $\Delta W^{(i-1)}$ can be written as:

$$\frac{\partial \Delta W^{(i-1)}}{\partial W^{(i-1)}} = \begin{cases} C_0 \frac{\partial^2 L}{\partial y^n \partial W^i}, & x^{(i-1)} > 0 \\ 0, & elsewise \end{cases} \tag{3.9}$$

Consider the back propagation with soft max activation and cross entropy loss. For the last layer, it can be shown that:

$$\hat{y}_j = \varsigma(x^N)_j = \frac{e^{x_j^N}}{\sum_{c=1}^{C} e^{x_c^N}}$$

where C is the number of neurons within last layer, also known as the number of classes for classification problem. Denote $\sum_{c=1}^{C} e^{x_c^N}$ as S. The derivative of above equation can be written as:

$$\frac{\partial \hat{y}_j}{\partial x_i^N} = \frac{\partial \frac{e^{x_i^N}}{S}}{\partial x_i^N} = \begin{cases} \frac{e^{x_i^N} S - e^{x_i^N} e^{x_i^N}}{S^2} = \hat{y}_j(1 - \hat{y}_j) & i = j \\ \frac{0 - e^{x_i^N} e^{x_i^N}}{S^2} = -\hat{y}_j \hat{y}_j & i \neq j \end{cases} \tag{3.10}$$

The cross entropy loss function $L(y, \hat{y})$ then is:

$$L(y, \hat{y}) = -\sum_{c=1}^{C} \delta_c \cdot \log(\hat{y}_c)$$

where δ_c is 1 if and only if sample belongs to class c. Therefore, for the last layer, we have

$$\frac{\partial L(y,\hat{y})}{\partial x_i^N} = -\sum_{c=1}^{C} \frac{\delta_c \partial \log(\hat{y}_c)}{\partial x_i^N} = -\sum_{c=1}^{C} t_j \frac{1}{\hat{y}_c} \frac{\partial \hat{y}_c}{\partial x_i^N}$$

By introducing equation 3.5 to above equation, we have:

$$\frac{\partial L(y,\hat{y})}{\partial x_i^N} = -y_i + \hat{y}_i$$

then equation 3.12 becomes:

$$\frac{\partial \Delta W_{l,m}^{(i-1)}}{\partial W_{l,m}^{(i-1)}} = \begin{cases} C_0 \sum_{c}^{C} \frac{\partial(-y_c + \hat{y}_c)}{\partial w_{l,m}^{i}}, & x_m^{(i-1)} > 0 \\ 0, & elsewise \end{cases} \tag{3.11}$$

As each layer's weight is only multiplied once during the forward pass, Equation 3.10 is either 0 or a constant, which implies that $\Delta W^i \propto W^i$.

Theorem 1 suggests that if a model's loss function has a linear derivative and is built with a piece-wise linear activation function, the gradient of each neuron is *linear* with respect to its current weight value. This implies that the weight W^i can be viewed as a scale for the learning rate. Specifically, with other layers frozen, given a training sample (x, y), the weight update can be written as:

$$\Delta W^i = C(x, y, W^i) \cdot \eta \tag{3.12}$$

where C is a function of the training sample and the current weight. By the nature of deep neural networks, the complexity of the function C increases as the model becomes deeper. For a network with piece-wise linear activations, according to Theorem 1, ΔW^i can be reformatted as $C'(x, y) \cdot W^i \cdot \eta$. In other words, the training of such networks depends primarily on the provided sample, making it more stable.

3.4 Illustrations

Following the theoretical analysis provided in the previous section, this section aims to empirically demonstrate how the weights of a neural network affect the

training dynamics. Specifically, a unit-wise analysis of a toy model is presented to understand the ability of networks with different activation functions to approximate an objective function.

3.4.1 Settings of Toy Model

This section uses a toy model to approximate a simple function, with the objective of understanding the behavior of each neuron during model convergence given different weight statuses. The model is a neural network with a 1D input, 1 hidden layer with 10 neurons, and a 1D output. The focus of this section is on ReLU networks and Sigmoid networks to better understand their performance differences. The parameters of the models are initialized from different distributions. To differentiate the model settings, the models are denoted as:

- $ReLU(w, b)$: A network with a ReLU activation function, where the weights and biases are initialized from a random uniform distribution $[-w, w]$ and $[-b, b]$, respectively.
- $Sigmoid(w, b)$: A network with a Sigmoid activation function, where the weights and biases are initialized from a random uniform distribution $[-w, w]$ and $[-b, b]$, respectively.

The objective function to be approximated by the toy model is

$$y = 2\cos(\pi x) + x(x-1) + 1. \tag{3.13}$$

During the training, an epoch of samples is set as the data pair $x_i, f(x_i)$, where x_i is an arithmetic sequence in $[-1, 1]$. All the models are trained with a learning rate of 0.001 until they reach convergence. There are two benefits to presenting this toy model:

- First, with a limited number of neurons, the objective function cannot be perfectly approximated. This limitation is more likely to expose potential issues in neural networks with different activation functions.

- Second, by illustrating the post-activation values, weight gradients, and bias gradients of each neuron, the training dynamics of neural networks with different activation functions can be compared.

For each model, the upper figure presents the following results:

- The ground truth value of the objective function $y(x)$, shown in red.
- The predicted value of the model $\hat{y}(x)$, shown in blue.
- The post-activation values $y_i^{(1)}$ of each neuron, depicted in different colors.

This means that

$$\hat{y}(x) = \sum_{i=1}^{10} y_i^{(1)}. \tag{3.14}$$

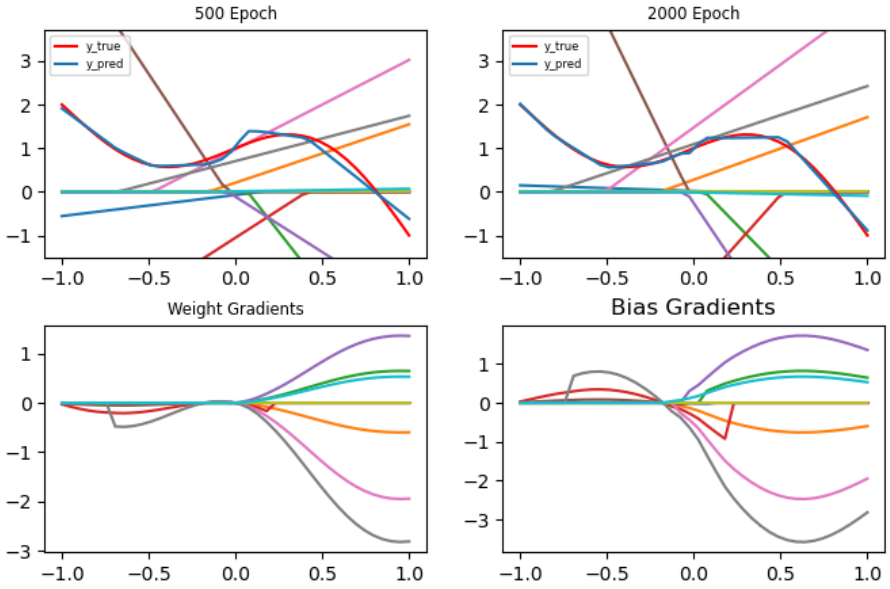
Similarly, the lower figure reports the neuron-wise gradient $\partial L(\hat{y}(x), y(x))/\partial W_i^{(1)}$ in different colors given the input x.

3.4.2 ReLU net vs Sigmoid net

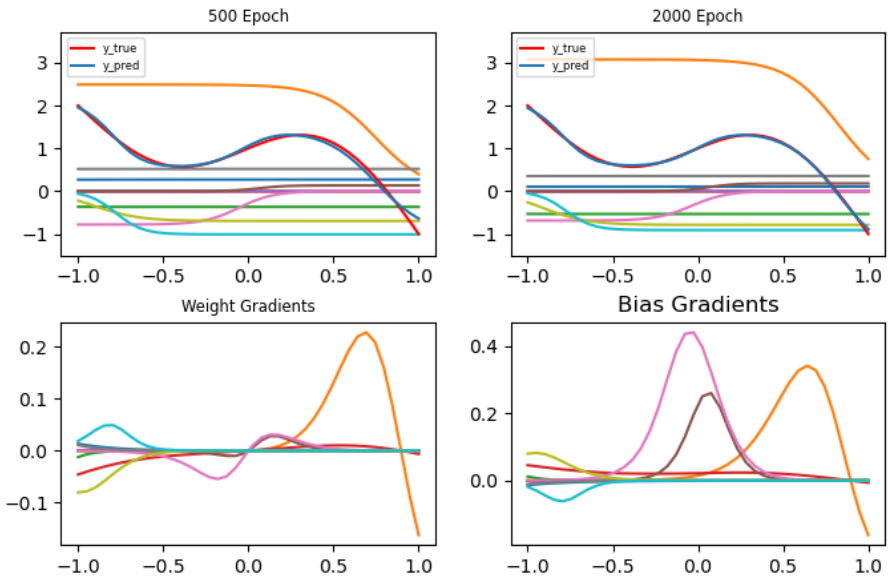
The experiment is designed to explore the performance of the ReLU net and Sigmoid net with varying weight initializations. Due to the different natures of the ReLU and Sigmoid activation functions, distinct parameter sets are used for each network. The selected parameter sets for the models are as follows:

- ReLU net: ReLU(15, 1), ReLU(5, 1), ReLU(1, 1), ReLU(1, 0),
- Sigmoid net: Sigmoid(50, 50), Sigmoid(15, 15), Sigmoid(15, 0), Sigmoid(1, 1)

These configurations are chosen to understand how different activation functions and initializations impact the training dynamics and performance of neural networks.



(a) ReLU(15, 1)



(b) Sigmoid(15, 15)

Figure 3.19: The performance and gradients of the dense networks with 1-d input, 1 hidden layer with 10 neurons, and 1-d output.

3.4.2.1 Best Performance Case

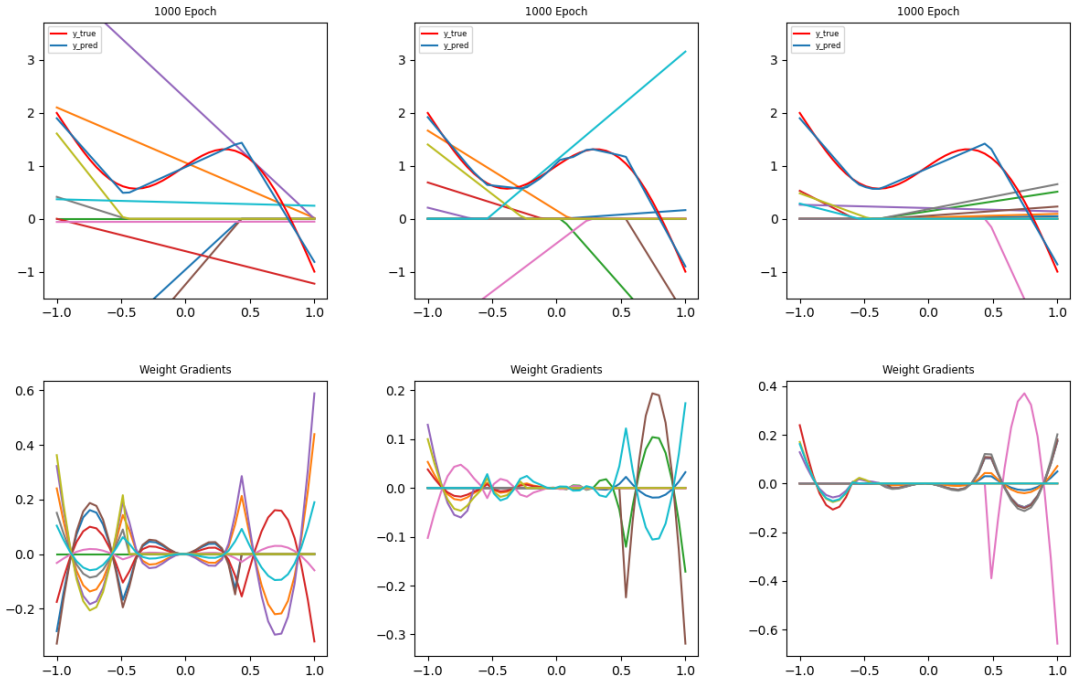
The investigation starts with comparing the models with the best performances. Figure 3.19 shows the performance of the model for ReLU(15, 1) and Sigmoid(15, 15). The upper parts illustrate the fitted curve after 500 and 2000 epochs, as well as the post-activation value of each neuron. The lower parts describe the gradients of W and b with respect to a single instance (x, y). Notice that because of the different forms between ReLU and Sigmoid functions, the breakpoints of ReLU(15, 1) and the center of Sigmoid(15, 15) are uniformly distributed around 0.

Figure 3.19(a) shows the result of $ReLU(15, 1)$. After 2000 epochs of training, the estimated function has a relatively small mean square error. However, there are several fitting gaps where the fitted curve is linear while the desired function is curvilinear, such as the region from $[0.2, 0.5]$ and $[0.6, 1]$. This is caused by the nature of the piece-wise linear activation function. As the toy model has only 10 neurons, it can only approximate the objective function with a piece-wise linear function with limited pieces.

Figure 3.19(b) presents the fitted curve of Sigmoid(15, 15), which shows better performance than its ReLU counterpart. However, one concerning finding is that the post-activation value of most neurons remains unchanged regardless of the input. In fact, only 3 neurons (orange, pink, and brown) are proactively contributing to the prediction, while the others can be viewed as constant.

The illustration in this section shows the neuron-level behavior in the simple model. It is noted that there are potential issues with both the ReLU net and the Sigmoid net. Intuitively, as model complexity increases, the ReLU net can create more piece-wise linear regions, allowing for a more precise approximation. However, for the Sigmoid net, since most neurons do not contribute to the prediction, this issue cannot be resolved merely by scaling the size of the network. This problem is referred to as the *dying neuron problem*. The next part of this section demonstrates that while the *dying neuron* issue is exacerbated when the Sigmoid net has poor initialization, it rarely affects the performance of the ReLU net.

3.4.2.2 Dying Neuron Issue

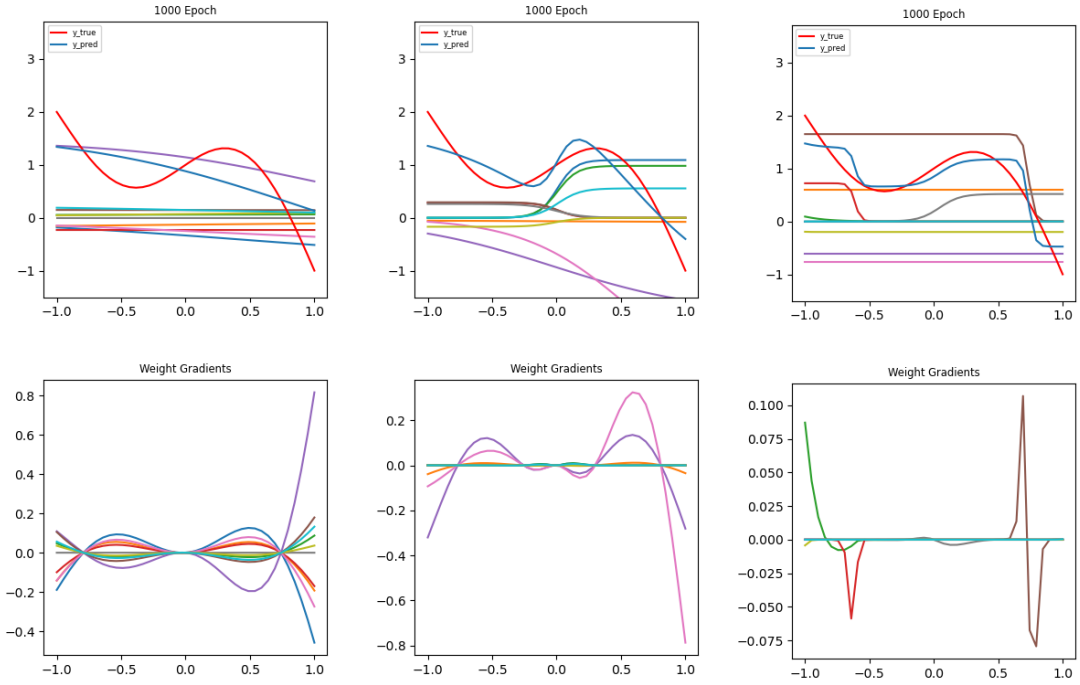


(a) ReLU(1, 1) (b) ReLU(15, 1) (c) ReLU(15, 0)

Figure 3.20: The performance and gradients of the model with ReLU activation. The upper graphs show the objective function, predicted results and the post-activation of each neuron of 1000 epoch training. In each of the figures, the red and blue lines represent the objective function and the predicted result, while each other line represents the post-activation (upper) and gradient (lower) value of a neuron.

Figure 3.20 and Figure 3.21 present results similar to those in Figure 3.19, but with different parameter sets. The upper graphs show the contribution of each neuron, the predicted value of the network, and the ground-truth value of the function. The predicted value and ground-truth value are shown in blue and red, respectively, while other curves represent the contribution of each neuron to the output. The lower graphs show the gradient of each neuron with respect to the input x.

Figures 3.20(a) to 3.20(c) display the results for ReLU(1, 1), ReLU(15, 1), and



(a) Sigmoid(1, 1) (b) Sigmoid(15, 0) (c) Sigmoid(50, 50)

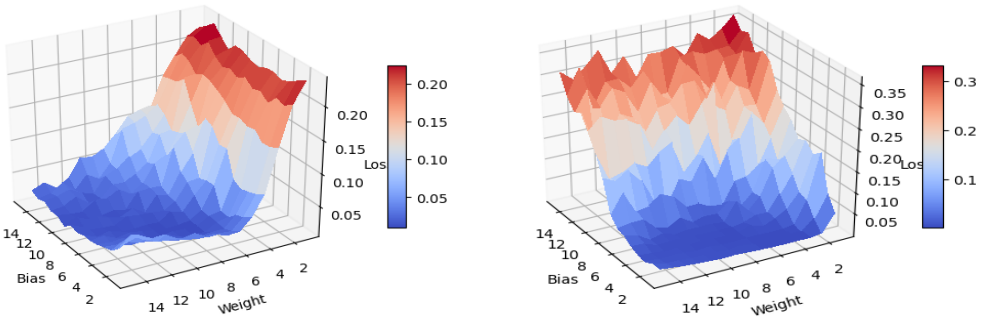
Figure 3.21: The performance and gradients of the model with the Sigmoid activation. The upper graphs show the objective function, predicted results and the post-activation of each neuron of 1000 epoch training. In each of the figures, the red and blue lines represent the objective function and the predicted result, while each other line represents the post-activation (upper) and gradient (lower) value of a neuron.

ReLU(15, 0). These figures reveal that the three models have similar performance regardless of initialization. This can be explained by the weight gradient graphs. Since ReLU is known for its asymmetry, it only responds to inputs x in half of its domain. Consequently, the gradients of ReLU are not neutralized by evenly distributed inputs within a defined domain. This means that the training of ReLU nets is less sensitive to weight initialization and more stable due to the asymmetry.

Among the three Sigmoid models, Sigmoid(1, 1), shown in Figure 3.21(a), has the worst performance, with almost all neurons yielding a constant regardless of the input. However, the gradient of the model with respect to an epoch of inputs

is non-zero. Due to the symmetry and limited range of the sigmoid activation function, the gradients of different samples cancel each other out. Therefore, instead of finding the global optimum, the training merely adjusts each neuron to a constant c such that $\int(f_i'(x) - c)dx \approx 0$, where f_i' is the gradient of neuron i. The only neuron that can provide useful information is colored purple due to its relatively high gradients around 1. Moreover, because the weight initialization is centered with less diversity, the backpropagation process fails to find the steepest direction for each input. Figures 3.21(b) and 3.21(c) show the performance of Sigmoid(15, 0) and Sigmoid(50, 50). Compared with Sigmoid(1, 1), Sigmoid(15, 0) and Sigmoid(50, 50) demonstrate better performances. However, the *dying neuron* issue still persists. For Sigmoid(15, 0), the small initialized bias causes most of the post-activation functions to be centered around zero, resulting in underfitting of the non-zero regions.

3.4.3 Search of Optimum Parameters



(a) Loss Mean of Sigmoid (b) Loss Mean of ReLU

Figure 3.22: Average Loss of model with different initialization value.

To better compare the effect of initialization on the model performance of ReLU and Sigmoid activations, the average performance of the models with different initializations is shown in Figure 3.22. For each set of initialization values (w, b), weights and biases are initialized from $U(-w, w)$ and $U(-b, b)$ respectively, where U denotes a random uniform distribution. The model is trained 50 separate times

for each set of parameters.

The average loss for the Sigmoid activation function forms a U-shaped surface centered around the line $w = b$. This indicates that the performance of the Sigmoid model is influenced by both weight and bias initialization. In contrast, the average loss for the ReLU activation function is an inclined plane where the loss is predominantly determined by bias initialization. In other words, to achieve the potential best performance of a Sigmoid net, more precise requirements for both weight and bias initialization need to be met.

3.5 Chapter Summary

This chapter serves as an introduction to the research undertaken in this dissertation, with the motivation of investigating the widely recognized performance gap among neural networks with different activation functions. The results reveal that a *dying neuron* issue is observed when the model becomes trapped in a local optimum. Similar to the gradient vanishing issue, the *dying neuron* issue also prevents the model from updating its weights. However, while the gradient vanishing issue arises due to overly saturated gradients, the *dying neuron* issue occurs because gradients from different samples can cancel each other out, making it difficult for the model to update. Moreover, once a neuron is trapped in a local optimum, it produces similar results regardless of the input. Due to the symmetry of the Sigmoid activation function, this issue is more pronounced in Sigmoid networks, contributing to the performance gap between ReLU and Sigmoid networks.

At the beginning of this chapter, Section 3.1 confirms the widely acknowledged statement that there are performance differences between neural networks with different activation functions. To understand the performance gap between neural networks with different activation functions, Section 3.2 re-investigates the *vanishing gradient* issue by monitoring pre-activation values, weights, and gradients of networks at different layers. The results indicate that the *vanishing gradient*

issue has been addressed by more recent techniques. Additionally, an instability in gradients and the gradient-to-weight ratio is observed in networks with poorer performance.

Section 3.3 provides an explanation of these observations by presenting a theoretical analysis of the training dynamics of neural networks. By examining the learning dynamics of deep networks, it is shown that for networks with Sigmoid and Tanh activations, the training is heavily influenced by the current status of the weights. The parameters of these networks tend to oscillate, preventing the network from reaching its optimum.

Section 3.4 bridges the theoretical investigation with empirical results by illustrating the weights and gradients of a toy model. This experiment aims to show how weights affect model performance by studying the behavior of individual neurons in networks with different weight initializations. It is found that, due to the symmetry of the Sigmoid and Tanh activation functions, once a neuron is trapped in a local optimum region, weight updates tend to cancel each other out, making it difficult for the neuron to escape this region.

With several introductory experiments in this Chapter, it is shown that there still exists performance difference between neural networks with different activation functions after the use of batch normalization and weight initialization. This suggests that it is worth to further investigate the performance of neural network from activation function perspective.

Chapter 4

Activation Pattern, Path and the Framework

This chapter introduces the analytic framework of this dissertation. The theoretical basis of this work is developed from the analysis of the activation region that was initially proposed for explaining the performance of neural network with a piece-wise linear activation function. The input domain of such a neural network $\mathcal{N}$ is separated into many regions, within each of the region the mapping of $\mathcal{N}$ is piece-wise linear. Previous research of this field includes the expressive ability, sensitivity, and potential issues of the network [170], [182]. However, there are several research gaps in the existing literatures, which are discussed and addressed by this dissertation as followed.

- First, the definition of activation regions is based on neural networks with piecewise linear activation functions but fails to consider other activation functions. This chapter generalizes the definition to accommodate other activation functions, allowing for a more appropriate investigation of neural network performance with different activation functions.

- Second, due to the complexity of neural networks, previous works began by studying the properties of single activation regions. To better understand

neural networks, it is necessary to describe the mapping relationships of the network across different activation regions. This chapter introduces *float neurons* and *fixed neurons* to describe the stability of neurons and to build connections between the mapping functions in different activation regions.

There are 5 sections in this chapter. Section 4.1 introduces the notations for neural network in this work. Section 4.2 introduces the concepts used in this dissertation, including the generalized activation pattern. As the cornerstone of the theoretical framework in this work, a special focus is placed on explaining the motivation and discussing the benefits of these concepts. In particular, the definition introduced in this section divides the domain of activation functions into regions and uses an indexed family to record the signs of neurons. This addresses the research gap where activation patterns could only be applied to piecewise linear functions and could not be extended to larger regions.

Section 4.3 generalizes the idea of describing neural network performance based on the status of neurons from a single activation region to a larger subspace. It begins by studying two activation regions that are separated by a bent hyperplane, referred to as adjacent activation regions, and then discusses the generalized cases.

After defining the main concepts and basic properties, Section 4.4 presents illustrations of the proposed concepts with the objective of aiding in the understanding of the definitions. Section 4.5 summarizes the framework.

4.1 Neural Network and Mapping Functions

The main objective of this dissertation is to investigate the interpretability of feedforward neural networks. Therefore, unless otherwise specified, neural networks in this work refer to feedforward neural networks designed to solve classification tasks.

Assume that the classification task has c different classes under the distribution $\mathbb{D} = \mathbb{D}_{\mathbf{x}} \times \mathbb{D}_y$, where $\mathbb{D}_{\mathbf{x}}$ and $\mathbb{D}_y$ are the distributions of observations $\mathbf{x}$ and labels

y for the samples. Denote the support of $\mathbb{D}_{\mathbf{x}}$ and $\mathbb{D}_y$ are $\mathbb{R}^n \times \{1, 2, ..., c\}$, which means that:

$$\mathbf{x} \in \mathbb{R}^n, \quad y \in \{1, 2, ..., c\}, \tag{4.1}$$

For every $(\mathbf{x}, y) \sim \mathbb{D}$. In other words, given $(\mathbf{x}, y)$ sampled from the joint distribution, the observed data $\mathbf{x}$ is a variable from the $\mathbb{R}^n$ space, while y is the label of the data. Under the trivial assumption that the data distribution $\mathbb{D}_{\mathbf{x}}$ has no atoms, this means that for any $\mathbf{x}_0 \in \mathbb{R}^n$, the probability $\mathbb{P}(\mathbf{x} = \mathbf{x}_0)$ is zero:

$$\mathbb{P}(\mathbf{x} = \mathbf{x}_0) = 0, \forall \mathbf{x}_0 \in \mathbb{R}^n. \tag{4.2}$$

The classification task aims to build a model to predict the label based on the observation $\mathbf{x}$. To address this task, the datasets D_{train}, D_{val}, and D_{test} are drawn from the data distribution for training, validating, and testing the model.

In this work, the model is a feedforward neural network $\mathcal{N}$. Denote π as the activation function of the network, and let θ be the parameter set of the model. The parameter set is assumed to have measure zero with respect to the Lebesgue measure. Let $f : \mathbb{R}^n \to \mathbb{R}^c$ be the mapping function of $\mathcal{N}$. Therefore, for an observation from the data distribution $\mathbf{x} \sim \mathbb{D}_{\mathbf{x}}$, the neural network $\mathcal{N}$ computes an output $f(\mathbf{x}) \in \mathbb{R}^c$, where $\mathbb{R}^c$ is the output space with a dimension of c. The label of $\mathbf{x}$ is predicted as $\hat{y} = \arg\max_{i \in 1,2,...,c} f_i(\mathbf{x})$.

Assume that the network has d blocks: $f = h_d \circ h_{d-1} \circ \cdots \circ h_1$. For $i \in \{1, 2, \ldots, d-1\}$, each block is defined as $h_i = \pi_i \circ \psi_i \circ \phi_i$, where π_i, ψ_i and ϕ_i are the activation function, batch-normalization layer and linear affine of layer i, respectively. The last layer $h_d = \psi_d \circ \phi_d$ omits the activation function. The intermediate outputs depend on both the input and model parameters, therefore can be viewed as functions. Given input $\mathbf{x}$, the input, output and pre-activation of block h_i are then represented as $x^{(i)}(\mathbf{x}; \boldsymbol{\theta}), y^{(i)}(\mathbf{x}; \boldsymbol{\theta})$ and $z^{(i)}(\mathbf{x}; \boldsymbol{\theta})$. This implies that the input and output of the model then can be denoted as $\mathbf{x} = x^{(1)}(\mathbf{x}; \boldsymbol{\theta})$ and $f(\mathbf{x}) = y^{(d)}(\mathbf{x}; \boldsymbol{\theta})$. Denote the

Data Distribution and Spaces	
$(\mathbf{x}, y)$	Data, label pair draw joint distribution $\mathbb{D}_x \times \mathbb{D}_y$.
$\mathbb{D}_{\mathbf{x}}$	Data distribution of input x.
$\mathbb{D}_y$	Data distribution of output y.
$\mathbb{R}^n$	Input space with n dimensions.
$\mathbb{R}^c$	Output space with c dimensions.
$\mathcal{R}$	An subspace in input space $\mathbb{R}^n$
Network Input, Output and Pre-activation	
$\mathcal{N}$	Neural network.
f	Mapping function of network.
θ	Parameter set of network $\mathcal{N}$.
h_i	The i-th block in neural network $\mathcal{N}$: $h_i = \pi_i \circ \psi_i \circ \phi_i$.
ϕ_i	The linear affine of layer i.
ψ_i	Batch-normalization of layer i.
π_i	Activation function of layer i.
$W^{(i)}$	The equivalent matrix of $\psi_i \circ \phi_i$.
$\beta^{(i)}$	The equivalent shift parameter of ψ_i.
$x_j^{(i)}(\mathbf{x}; \theta)$	The j-th input of i-th block given x and parameter θ.
$y_j^{(i)}(\mathbf{x}; \theta)$	The j-th output of i-th block given x and parameter θ.
$z_j^{(i)}(\mathbf{x}; \theta)$	The j-th pre-activation value of i-th block given x and parameter θ.

Table 4.1: Notations of Data Distribution and Neural Network

dimension of output of each layer as n^i, which implies:

$$\begin{aligned} y^{(i)}(\mathbf{x}; \boldsymbol{\theta}) &\in R^{n^i} \\ z^{(i)}(\mathbf{x}; \boldsymbol{\theta}) &\in R^{n^i} \\ x^{(i)}(\mathbf{x}; \boldsymbol{\theta}) &\in R^{n^{i-1}}. \end{aligned} \tag{4.3}$$

Figure 4.1 illustrates the input, pre-actiavtion and output for a blcok. For

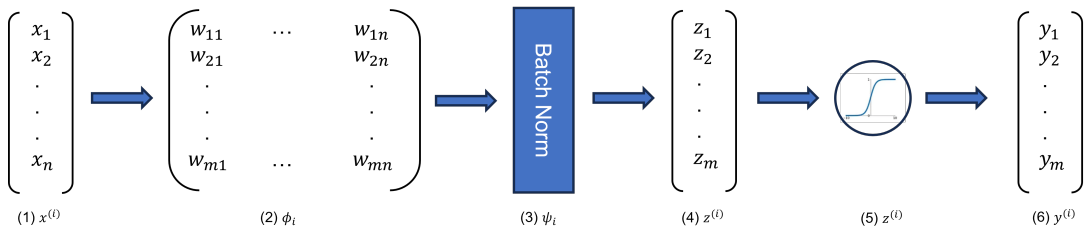


Figure 4.1: An illustration feedforward neural network block. (1): Input of layer i $x_j^{(i)}(\mathbf{x};\theta)$; (2) linear affine ϕ_i (3) batch normalization layer ψ_i (4) pre-activation $z_j^{(i)}(\mathbf{x};\theta)$ (5) activation function π_i (6) block output $y_j^{(i)}(\mathbf{x};\theta)$

reading continence, when the parameter set is fixed, $x^{(i)}(\mathbf{x};\boldsymbol{\theta}), y^{(i)}(\mathbf{x};\boldsymbol{\theta})$ and $z^{(i)}(\mathbf{x};\boldsymbol{\theta})$ are abbreviated as $x^{(i)}(\mathbf{x})$, $y^{(i)}(\mathbf{x})$ and $z^{(i)}(\mathbf{x})$.

Since the only nonlinearity in a feedforward neural network is provided by the activation function, the transformation of block i's input, $\mathbf{x}^{(i)}$, to the pre-activation of block i can be written as a linear affine transformation. In other words, the composition of $\psi_i \circ \phi_i$ can be represented by an equivalent matrix $W^{(i)} \in \mathbb{R}^{n^{i-1} \times n^i}$ and an equivalent shift vector $\beta^{(i)} \in \mathbb{R}^{n^i}$:

$$z^{(i)}(\mathbf{x}) = \psi_i \circ \phi_i(x^{(i)}(\mathbf{x})) = W^{(i)} x^{(i)}(\mathbf{x}) + \beta^{(i)}. \tag{4.4}$$

Table 4.1 summarizes the notations introduced in this section. Throughout this dissertation, the notations for the neural network and the data distribution will remain unchanged unless otherwise specified.

4.2 Activation Pattern / Region

This section defines the basic concepts of this dissertation. The research undertaken in this dissertation revolves around the activation functions of neural networks. In particular, the framework proposed in this work is developed from the activation pattern of the piecewise linear activation function.

As the starting point of the theoretical analysis in this research, this section first introduces a generalized definition of the activation pattern, focusing on the

improvements proposed in this work as well as the motivation behind them. Next, an explanation of the activation pattern and activation region is presented to provide a better understanding of the relationship between them. Finally, several properties of the mapping function within an activation region, including convexity, continuity, and Lipschitz properties, are discussed.

4.2.1 Definition

Given a neural network $\mathcal{N}$ with a piecewise linear activation, the input domain is partitioned into numerous regions. Within each region $\mathcal{R} \subset \mathbb{R}^n$, the mapping function $f : \mathcal{R} \to \mathbb{R}^c$ is linear, meaning that:

$$f(x) - f(x') = k(x - x'), \forall x, x' \in \mathcal{R}. \tag{4.5}$$

Within each of these linear regions, the linearity of each neuron remains unchanged if the region is convex. Based on this observation, the input space can be decomposed into numerous linear regions. Each region can be described by assigning each neuron a sign that indicates its activation status. The region and the corresponding set of signs are then referred to as the activation region and activation pattern, as defined in previous works [19], [170]. In this work, this definition is generalized and restated as follows.

- The definition proposed in this work removes the piecewise linear constraint for the activation functions. Instead of separating the activation function according to piecewise linearity, it uses a set of breakpoints to partition the domain of the activation function, which enables the investigation of neural networks with continuous activation functions.

- In addition to assigning each neuron a sign, the generalized definition introduces an indexed family to label each neuron. This allows tracking the status of neurons across a dataset. In particular, as the neurons are indexed, the activation patterns of different data points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$ are comparable,

which helps to understand how the mapping function changes for different data.

Definition 1 formally states above observation and extends it to a general activation.

Definition 1 (Generalized Activation Pattern / Region)**.** *Let $\mathcal{N}$ be a feed forward neural network defined as in Section 4.1. A set of breakpoints $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_q\}$ separates the domain of activation function π into $q+1$ intervals $U = \{U_0, U_1, \ldots, U_q\}$, where*

$$U_i = \begin{cases} (-\infty, \gamma_1), & i = 0, \\ (\gamma_i, \gamma_{i+1}), & i = 1, \ldots, q-1, \\ (\gamma_q, +\infty), & i = q. \end{cases}$$

An activation pattern of $\mathcal{N}$ is an indexed family

$$\mathcal{A} := \{a_j^{(i)} \ | a_j^{(i)} \in \{0, 1, \ldots, q\}, (i,j) \in \mathcal{I}\}. \tag{4.6}$$

Given a pattern $\mathcal{A}$, the corresponding activation region is defined as:

$$\mathcal{R}(\mathcal{A}; \theta, \pi, \Gamma) := \{\mathbf{x} \in \mathbb{R}^n | z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) \in U_{a_j^{(i)}}, a_j^{(i)} \in \mathcal{A}\} \tag{4.7}$$

where $z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta})$ is the pre-activation value of j-th element in layer i. Conversely, for any $\mathbf{x} \in R^n$, the pattern of $\mathbf{x}$ is denoted as:

$$\hat{\mathcal{A}}(\mathbf{x}; \boldsymbol{\theta}, \pi, \Gamma) = \{\hat{a}_j^{(i)}(\mathbf{x}) | (i,j) \in \mathcal{I}\} \tag{4.8}$$

where:

$$\hat{a}_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}, \pi, \Gamma) = \begin{cases} k, & z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) \in U_k, \\ \textit{abstained}, & z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) \in \{\gamma_1, \gamma_2, \ldots, \gamma_q\} \end{cases} \tag{4.9}$$

Moreover, if π is continuous within every interval U_i, the Γ is a continuous separation of the network $\mathcal{N}$.

4.2.2 Understanding Activation Region / Pattern

To understand the concept of an activation pattern, the start point of this section is Equation 4.7, which is restated below:

$$\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma) := \{\mathbf{x}\in\mathbb{R}^n | z_j^{(i)}(\mathbf{x};\boldsymbol{\theta}) \in U_{a_j^{(i)}}, a_j^{(i)} \in \mathcal{A}\}$$

Given a network $\mathcal{N}$ and an input space $\mathbb{R}^n$, the generalized activation pattern $\mathcal{A}$ describes the activation status of each unit in the intermediate layers. Figure 4.2 demonstrates how the Sigmoid and ReLU activation are separated by breakpoints $\{-3,3\}$ and $\{0\}$. Given $\mathbf{x}\in\mathbb{R}^n$, the pre-activation of the j-th neuron in layer i is denoted as $z_j^{(i)}(\mathbf{x};\boldsymbol{\theta})$. Consider $\mathcal{N}$ has Signmoid activation function, if the $z_j^{(i)}(\mathbf{x};\boldsymbol{\theta}) < -3$, then the pattern of neuron (i,j) for input $\mathbf{x}$ is 0:

$$\hat{a}_j^{(i)}(\mathbf{x};\boldsymbol{\theta},\pi,\Gamma) = 0. \tag{4.10}$$

Similarly, the pattern of (i,j) is then 1 or 2 if $z_j^{(i)}(\mathbf{x};\boldsymbol{\theta})$ is in $(-3,3)$ or $(3,\infty)$, respectively. For every $\mathbf{x}\in\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma)$ in the corresponding activation region, the pre-activation value of each neuron $z^{(i)}j(\mathbf{x};\theta)$ lies within the $a^{(i)}j$-th interval of the activation π domain. This means that the activation region can be viewed as an operator $\mathcal{R}(\cdot)$ that maps an indexed family $\mathcal{A}$ to a subspace $\mathcal{R}\subset\mathbb{R}^n$. It identifies the region $\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma)$ such that

$$\mathbf{z}_j^{(i)}(\mathbf{x}) \in U_{a_j^{(i)}}, \forall\mathbf{x}\in\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma),$$

where $\mathbf{z}^{(i)}j(\mathbf{x})$ is the pre-activation value of neuron (i,j), $Ua^{(i)}j$ is the $a^{(i)}j$-th interval from the separation Γ, and $a_j^{(i)}\in 0,1,\ldots,q$ is the pattern of neuron (i,j).

To better elaborate the activation pattern of a neuron, Figure 4.2 demonstrates how the Sigmoid and ReLU activation are separated by breakpoints $\{-3,3\}$ and $\{0\}$. Given input $\mathbf{x}$ and a neuron (i,j),

Based on previous works [170][253], Equation 4.8 further introduces an inverse operator $\hat{\mathcal{A}}(x;\theta,\pi,\Gamma)$ that computes the activation pattern of an input $\mathbf{x}\in\mathbb{R}^n$ in Definition 1. This allows for the study of the robustness of the network by

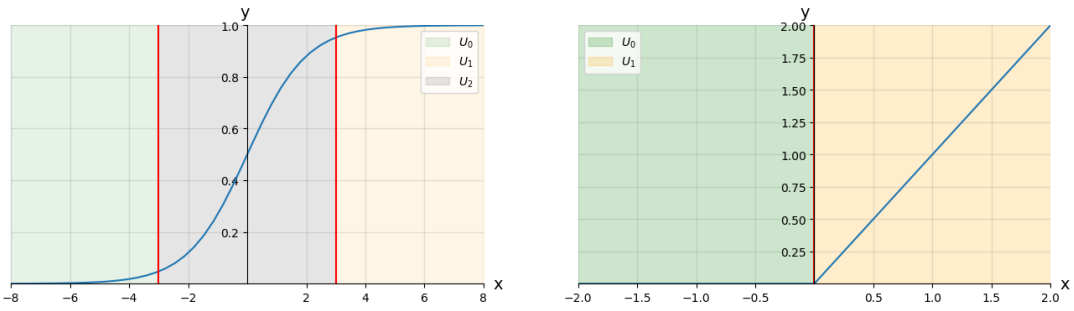


Figure 4.2: An illustration of how a set of breakpoints separate the domain of an activation function into different regions.

investigating the stability of the activation pattern in the sphere $B_p(\mathbf{x}, r)$. When dependencies are fixed, Equations 4.7 and 4.8 are abbreviated as $\mathcal{R}(\mathcal{A})$ and $\hat{\mathcal{A}}(x)$.

With a set of breakpoints, the activation function can be split into several parts according to its mapping relationship. For example, the set 0 separates ReLU into activated and deactivated regions, while the set $-1, 1$ separates tanh into two semi-constant regions at the sides and a semi-linear region around 0. When dependencies are fixed, Equations 4.7 and 4.8 are abbreviated as $\mathcal{R}(\mathcal{A})$ and $\hat{\mathcal{A}}(x)$. At the same time, the breakpoints and the network $\mathcal{N}$ define a set of hyperplanes:

$$H_{ijk} := \{\mathbf{x} | z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) = \gamma_k\}, i \in [d], j \in [n_i], k \in [q], \tag{4.11}$$

where $[n] = 1, 2, \ldots, n$ and n_i is the output size of block i. The input space is then divided into numerous connected components by H_{ijk}. For a given region $\mathcal{R}$, the post-activations of each neuron for any $\mathbf{x}$ and $\mathbf{x}' \in \mathcal{R}$ are within the same interval, which means that:

$$(z_j^{(i)}(\mathbf{x}) - \gamma_k)(z_j^{(i)}(\mathbf{x}') - \gamma_k) > 0, \forall \mathbf{x}, \mathbf{x}' \in \mathcal{R}, \forall i, j, k$$

This suggests that the non-linearity of each neuron within a region $\mathcal{R}$ is similar. Moreover, the activation pattern identifies the status of each neuron and provides a way to describe the mapping function in each region.

4.2.3 Regions and bent hyperplanes

Given a neural network $\mathcal{N}$, the input space is partitioned into numerous connected components $\mathcal{R}$ by a set of hyperplanes defined in Equation 4.11:

$$H_{ijk} := \{\mathbf{x} | z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) = \gamma_k\}, i \in [d], j \in [n_i], k \in [q],$$

where each of the hyperplanes is defined by neuron (i, j) and the k-th breakpoint. The activation pattern of data in each of the connected components $\mathcal{R}$ is the same. This means that the non-linearity of the neural network $\mathcal{N}$ is constrained by the activation pattern of neurons within the region $\mathcal{R}$. In particular, if the activation function of network $\mathcal{N}$ is piecewise linear, then the neural network $\mathcal{N}$ behaves as a linear function in region $\mathcal{R}$.

In Definition 1, each neuron is assigned a pattern based on its pre-activation value. Given a set of breakpoints Γ and a data point $x \in \mathbb{R}^n$, the post-activation value of each neuron is either within an interval $z^{(i)}j(\mathbf{x}; \boldsymbol{\theta}) \in U_k$ defined by its activation pattern or equals one of the breakpoints $z^{(i)}j(\mathbf{x}; \boldsymbol{\theta}) \in \Gamma$:

$$\hat{a}_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}, \pi, \Gamma) = \begin{cases} k, & z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) \in U_k, \\ \text{abstained}, & z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) \in \{\gamma_1, \gamma_2, \ldots, \gamma_q\} \end{cases} \tag{4.12}$$

The first case indicates that the post-activation value of neuron (i, j) for input $\mathbf{x}$ is in the interval $U_k, k \in 0, 1, \ldots, k$. In this case, a sign k will be assigned to this neuron for data $\mathbf{x}$. The latter case, on the other hand, suggests that $\mathbf{z}^{(i)}j(\mathbf{x}, \boldsymbol{\theta}) = k$. This means that neuron (i, j) satisfies Equation 4.11 at breakpoint k for input $\mathbf{x}$, and therefore $\mathbf{x}$ is located on the bent hyperplane $Hijk$.

To conclude the above two cases, given any data $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}$ is either located within an activation region or on a bent hyperplane. In other words, the input space can be decomposed into a set of activation regions and bent hyperplanes. This is described by the following lemma.

Lemma 1 (Decompose Input Space)**.** *Let $\mathcal{N}$ be a feedforward neural network as defined in Section 4.1 with a monotonic activation function π. Given Γ as a*

continuous separation of network $\mathcal{N}$, the input space can be expressed as:

$$\mathbb{R}^n = \left[\bigcap_{\forall A} \mathcal{R}(\mathcal{A}; \boldsymbol{\theta}, \pi, \Gamma)\right] \bigcup \left[\bigcap_{\forall i,j,k} H_{ijk}(\boldsymbol{\theta})\right] \tag{4.13}$$

where $\mathcal{R}(\mathcal{A}; \boldsymbol{\theta}, \pi, \Gamma)$ is the activation region of pattern $\mathcal{A}$, H_{ijk} is the bent-hyperplane defined as Equation 4.11.

Proof (Lemma 1)**.** *Lemma 1 can be proved directly according to the definition. For any $\mathbf{x} \in \mathbb{R}^n$, the pre-activation value for each neuron (i, j) is either:*

- *locates within an interval U_k*
- *equals to one of the breakpoints q_k.*

For the first case, without loss of generality, assume that $z_j^{(i)}(\mathbf{x}; \boldsymbol{\theta}) \in U_{a_j^{(i)}}$, then the pattern of neuron (i, j) is $a_j^{(i)}$. If every neuron satisfies the first case on data $\mathbf{x}$, then every neuron can be assigned a pattern accordingly. The activation pattern of $\mathbf{x}$ is then defined as:

$$\mathcal{A} := \{a_j^{(i)} | (i, j) \in \mathcal{I}\},$$

therefore $\mathbf{x}$ is in the activation region $\mathcal{R}(\mathcal{A}; \boldsymbol{\theta}, \pi, \Gamma)$.

Otherwise, if there exists m neurons $\{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$ satisfies the second case, then assume that :

$$z_{j_l}^{(i_l)}(\mathbf{x}; \boldsymbol{\theta}) = \gamma_{k_l}, l = 1, 2, \ldots, m.$$

This means that $\mathbf{x}$ is on the intersection of a set of bent-hyperplanes $\{H_{i_l j_l k_l}\}$.

Summarizing above, for every input $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}$ is either in an activation region or on a bent-hyperplane:

$$\mathbf{x} \in \left[\bigcap_{\forall A} \mathcal{R}(\mathcal{A}; \boldsymbol{\theta}, \pi, \Gamma)\right] \bigcup \left[\bigcap_{\forall i,j,k} H_{ijk}(\boldsymbol{\theta})\right], \forall \mathbf{x} \in \mathbb{R}^n.$$

Moreover, by definition, the activation region and bent-hyperplane are defined in the $\mathbb{R}^n$. This means that:

$$\mathbf{x} \in \mathbb{R}^n, \forall \mathbf{x} \in \left[\bigcap_{\forall A} \mathcal{R}(\mathcal{A}; \boldsymbol{\theta}, \pi, \Gamma)\right] \bigcup \left[\bigcap_{\forall i,j,k} H_{ijk}(\boldsymbol{\theta})\right].$$

This leads to the conclusion of Lemma 1.

Under the trivial assumptions in Section 4.1, the data distribution has no atoms, and the parameter set of the neural network $\mathcal{N}$ has zero measure with respect to the Lebesgue measure. In other words, given $\mathbf{x} \sim \mathbb{D}_x$, Equation 4.11 holds with probability 0 for any i, j, k, as described by Lemma 2.

Lemma 2 (Within Activation Region)**.** *Let $\mathcal{N}$ be a feed forward neural network defined as in Section 4.1 with piece-wise linear function π. Assume that the data distribution $\mathbb{D}$ has no atom:*

$$P(\mathbf{x} = \mathbf{x}_0, y = y_0) = 0, \forall (\mathbf{x}_0, y_0) \in \mathbb{R}^n \times \mathbb{R}^c,$$

then given $\mathbf{x} \in \mathbb{R}^n$, almost surely, there exists an activation pattern $\mathcal{A}$ such that $\mathbf{x} \in \mathcal{R}(\mathcal{A})$. In other words, the probability of $\mathbf{x}$ locates on a bent-hyperplane is 0:

$$P(z_j^{(i)}(\mathbf{x}) = \gamma_k) = 0, \forall i, j, k. \tag{4.14}$$

Proof (Lemma 2)**.** *The first step in proving Lemma 2 is to measure the volume of the hyperplane H_{ijk} in the $\mathbb{R}^n$ space. Let $\mathcal{I}$ be the index set of the network $\mathcal{N}$, as defined in Equation A.1. Denote $\mathcal{I}^C := \mathcal{I} \backslash (i, j) \subset \mathcal{I}$ as the set of neurons that excludes neuron (i, j). Given such a dataset, consider $\mathcal{A}_{\mathcal{I}^C}$ to be an incomplete activation pattern, which defines an incomplete activation region $\mathcal{R}(\mathcal{A}_{\mathcal{I}^C})$.*

The bent hyperplane H_{ijk} can be written as the union of all the segments:

$$H_{ijk} = \left(\bigcup_{\forall \mathcal{R}(\mathcal{A}_{\mathcal{I}^C})} (\mathcal{R}(\mathcal{A}_{\mathcal{I}^C}) \cap H_{ijk}) \right) \bigcup \left(\bigcup_{\forall i' \neq i, j' \neq j, k' \neq k} (H_{i'j'k'} \cap H_{ijk}) \right) \tag{4.15}$$

Using the fact that the Lebesgue measure is countably additive, Lemma 2 can be proved by showing that each of the intersections is of measure zero.

First, consider the former part of the above equation. Given an incomplete activation region, the mapping in region $\mathcal{R}(\mathcal{A}_{\mathcal{I}^C})$ is a linear function since the activation function π is piecewise linear. Therefore, the pre-activation function $\mathbf{z}_m^{(l)}(\mathbf{x})$ is also a linear function:

$$\mathbf{z}_m^{(l)}(\mathbf{x}) = < \mathbf{x} - b, \mathbf{a} >,$$

where $\mathbf{a}$ *and* $\mathbf{b}$ *are the scale and shift factor of the linear mapping. Let* $\mathcal{R}_m^{(l)}$ *be the subspace that satisfies the constraint for neuron* (l, m)*:*

$$\mathcal{R}_m^{(l)} := \{\mathbf{x} \in \mathbb{R}^n | \mathbf{z}_m^{(l)}(\mathbf{x}) \in U_{a_m^{(l)}}\},$$

where $\mathbf{z}_m^{(l)}(\mathbf{x})$ *is the pre-activation of neuron* (l, m)*,* $a_m^{(l)}$ *is the pattern of neuron* (l, m)*. The intersection between* $\mathcal{R}_m^{(l)}$ *with* H_{ijk} *is then:*

$$\mathcal{R}_m^{(l)} \cap H_{ijk} := \{\mathbf{x} \in \mathbb{R}^n | \mathbf{z}_m^{(l)}(\mathbf{x}) \in U_{a_m^{(l)}}, \mathbf{z}_j^{(i)} = \gamma_k\},$$

where γ_k *is the* k*-th breakpoint in* Γ*. Since* $\mathbf{z}_m^{(l)}(\mathbf{x}; \boldsymbol{\theta})$ *is linear in* $\mathcal{R}_m^{(l)}$*, it is also linear in the intersection* $\mathcal{R}_m^{(l)} \cap H_{ijk}$*. Therefore, showing that* $\mathcal{R}_m^{(l)} \cap H_{ijk}$ *is measure zero is equivalent to show that a hyperplane in a subspace of* $\mathbb{R}^n$ *is measure zero. Consider the case of* $\{x_n = 0\} \subset \mathcal{R}_m^{(l)}$*, which can be generalized by applying linear affine to the hyperplane. The hyperplane can be covered by a countable set of boxes at* $B_z = \bigcup z \in \mathbb{Z} \underbrace{[z, z+1] \times [z, z+1] \cdots \times [z, z+1]}_{n-1 \text{ dimension box}} \times [-h, h]_n$ *for arbitrary small* $h > 0$*. Let* $h \to 0$*, then the* B_z *is a measure zero set with respect to Lebesgue measure. Since the union of measure zero set is also measure zero set, then* $\mathcal{R}_m^{(l)} \cap H_{ijk}$ *has measure zero with respect to Lebesgue measure.*

Equation 4.7 suggests that the activation region is the intersection of all the subspaces $\mathcal{R}_m^{(l)}$*:*

$$\mathcal{R}(\mathcal{A}_{\mathcal{I}^C}) = \bigcup_{(l,m) \notin \mathcal{I}^C} \{\mathbf{x} \in \mathbb{R}^n | \mathbf{z}_m^{(l)}(\mathbf{x}) \in U_{a_m^{(l)}}\},$$

which means that each component of the former part in Equation 4.15 can be written as:

$$(\mathcal{R}(\mathcal{A}_{\mathcal{I}^C}) \bigcap H_{ijk}) = \bigcup_{(l,m) \in \mathcal{I}^C} (\mathcal{R}_m^{(l)}) \bigcup \cap H_{ijk} = \bigcup_{(l,m)} \left(\mathcal{R}_m^{(l)} \cap H_{ijk}\right).$$

The intersection of measure zero set is also measure zero, therefore $(\mathcal{R}(\mathcal{A}_{\mathcal{I}^C}) \bigcap H_{ijk})$ *is a measure zero set.*

Each component of the latter part of Equation 4.15 is the intersection of two bent-hyperplanes. Given bent-hyperplane H_{ijk} *and* $H_{i'j'k'}$*, a similar incomplete activation region can be constructed in the similar way as above by with the index set* $\mathcal{I}^C =$

$\mathcal{I}\backslash\{(i,j),(i',j')\}$. *For arbitrary incomplete activation pattern* $\mathcal{A}_{\mathcal{I}^C}$, *the intersection* $\mathcal{A}_{\mathcal{I}^C} \cap H_{ijk} \cap H_{i'j'k'}$ *is a hyperplane on* $n-2$ *dimension, therefore has measure 0 at* n *dimension. The union of all such intersection is also measure zero.*

The cumulative probability of $\mathbf{x}$ *on a bent-hyperplane can be calculated with Lebesgue integral:*

$$P(\mathbf{x} \in H) = \int_H P(\mathbf{x}) d\mu,$$

where $H := \bigcap_{\forall i,j,k} H_{ijk}$. *Since the data distribution* $\mathbb{D}$ *has no atom and* H *is the union of countable measure zero set, above equation equals to 0. Therefore, almost surely,* $\mathbf{x} \sim \mathbb{D}_{\mathbf{x}}$ *locates within an activation region instead of bent-hyperplane.*

The above proof uses the incomplete activation pattern, which is introduced in Definition 3 and discussed in Section 4.3. Since the probability of a data point $\mathbf{x}$ being located on a bent hyperplane is zero, Lemma 2 implies that, almost surely, every input $\mathbf{x} \in \mathbb{R}^n$ is in an activation region. This ensures that in studying the interpretability of neural networks, presuming that the activation pattern of $\mathbf{x}$ exists is trivial and will not affect the results.

4.2.4 Convexity

Previous works suggest that for a network with piecewise linear activation and a parameter set θ of measure zero with respect to the Lebesgue measure, the linear activation regions are convex [170]. In this section, it is claimed that the convexity of activation regions holds for any monotonic activation function.

Lemma 3 (Convexity)**.** *Let* $\mathcal{N}$ *be a feed forward neural network defined as in Section 4.1. If the activation function* π *is monotonic, then for any activation pattern* $\mathcal{A}$, *the corresponding activation region* $\mathcal{R}(\mathcal{A})$ *is convex.*

Proof (Lemma 3)**.** *This proof starts with the transformation within each block. Given the* i*-th block* h_j *of the neural network, denote the input space and output space of the network as* $\mathbb{R}^{n_{i-1}}$ *and* $\mathbb{R}^{n_i}$, *where* n_{i-1} *and* n_i *are the sizes of the outputs*

for block $i-1$ and block i. Each data point $\mathbf{x}^{(i)} \in \mathbb{R}^{n_{i-1}}$ from the input space of the i-th layer is mapped to the pre-activation $\mathbf{z}^{(i)} \in \mathbb{R}^{n_i}$ with a linear affine transformation:

$$\mathbf{z}^{(i)} = W^{(i)}\mathbf{x}^{(i)} + \beta^{(i)},$$

where $W^{(i)}$ and $\beta^{(i)}$ are the equivalent weights and bias of layer i. The output of this block $y^{(i)}$ is computed by applying an element-wise non-linearity π to the pre-activation value $\mathbf{z}^{(i)}$:

$$y^{(i)} = \left(\pi(\mathbf{z}_1^{(i)}), \pi(\mathbf{z}_2^{(i)}), \ldots, \pi(\mathbf{z}_{n_i}^{(i)})\right)^T$$

Given an activation pattern $\mathcal{A}$, let $S_j^{i-1} \subset \mathbb{R}^{n_{i-1}}$ be the subspace in $\mathbb{R}^{n_{i-1}}$ such that every $\mathbf{x}^{(i)} \in S_j^{i-1}$ satisfies the condition of neuron (i, j):

$$S_j^{i-1} := \{\mathbf{x}^{(i)} \in R^{n_{i-1}} | \left[W^{(i)}\mathbf{x}^{(i)}\right]_j + \beta_j^{(i)} \in U_{\hat{a}_j^{(i)}}, j = 1, 2, \ldots, n_i\},$$

where $\hat{a}_j^{(i)}$ is the activation pattern of neuron (i, j), $U_{\hat{a}_j^{(i)}}$ is the $\hat{a}_j^{(i)}$-th interval as per defined in Definition 1 and n_i is the output size of layer i.

Given that the mapping from $\mathbf{x}^{(i)}$ to $\mathbf{z}^{(i)}$ is a linear affine transformation and the interval of the activation function $U_{\hat{a}_j^{(i)}}$ is convex, the region S_j^{i-1} is also convex. The intersection of all the $S^{i-1}j$ defines a subspace $S^{i-1} \subset \mathbb{R}^{ni-1}$:

$$S^{i-1} = \bigcup_{j=0}^{n_i} S_j^{i-1}$$

such that every $\mathbf{x} \in S^{i-1}$, the constraint of neuron (i, j) is satisfied:

$$\left[W^{(i)}\mathbf{x}^{(i)}\right]_j + \beta_j^{(i)} \in U_{\hat{a}_j^{(i)}}, \forall \mathbf{x}^{(i)} \in S^{i-1}, \forall j \in [n_i].$$

As the intersection of convex space is also convex, S^{i-1} is convex.

The input of block i is the output of block $i-1$. Let P^{i-1} be a subspace in $\mathbb{R}^{n_{i-1}}$ such that for every pre-activation of $i-1$-th block $\mathbf{z}^{(i-1)} \in R^{n_{i-1}}$, the block output $y^{(i)}$ is in S^{i-1}:

$$P^{i-1} = \left\{\mathbf{z}^{(i-1)} \in R^{n_{i-1}} | \left(\pi(\mathbf{z}_1^{(i-1)}), \pi(\mathbf{z}_2^{(i-1)}), \ldots, \pi(\mathbf{z}_{n_i}^{(i-1)})\right)^T \in S^{i-1}, \forall j \in [n_{i-1}]\right\}.$$

Given $\mathbf{z}^{(i-1)}, \mathbf{z}'^{(i-1)} \in P^{i-1}$ *and* $\lambda \in (0,1)$*, the following proves that* $\mathbf{z}''^{(i-1)} = \lambda\mathbf{z}^{(i-1)} + (1-\lambda)\mathbf{z}'^{(i-1)} \in P^{i-1}$*. For any* $j \in [n_{i-1}]$*, without loss of generality, assume that* $\mathbf{z}_j^{(i-1)} \leq \mathbf{z}_j'^{(i-1)}$*, then* $\mathbf{z}_j^{(i-1)} \leq \mathbf{z}_j''^{(i-1)} \leq \mathbf{z}_j'^{(i-1)}$*. Since* π *is monotonic, it can be shown that*

$$\pi(\mathbf{z}_j^{(i-1)}) \leq \pi(\mathbf{z}_j''^{(i-1)}) \leq \pi(\mathbf{z}_j'^{(i-1)}). \tag{4.16}$$

Let

$$\mathbf{x}^{(i)} = \left(\pi(\mathbf{z}_1^{(i-1)}), \pi(\mathbf{z}_2^{(i-1)}), \ldots, \pi(\mathbf{z}_{n_i}^{(i-1)})\right)$$
$$\mathbf{x}'^{(i)} = \left(\pi(\mathbf{z}_1'^{(i-1)}), \pi(\mathbf{z}_2'^{(i-1)}), \ldots, \pi(\mathbf{z}_{n_i}'^{(i-1)})\right)$$
$$\mathbf{x}''^{(i)} = \left(\pi(\mathbf{z}_1''^{(i-1)}), \pi(\mathbf{z}_2''^{(i-1)}), \ldots, \pi(\mathbf{z}_{n_i}''^{(i-1)})\right)$$

Given that $\mathbf{x}^{(i)}, \mathbf{x}'^{(i)} \in S_j^{i-1}$ *and Equation 4.16,* $\mathbf{x}_j''^{(i)} \in U_{\hat{a}_j^{(i)}}$*, therefore* $\mathbf{x}''^{(\in)} S_j^{i-1}$*. By generalizing the result to every* $j \in [n_j]$*, it can be shown that:*

$$\left(\pi(\mathbf{z}_1''^{(i-1)}), \pi(\mathbf{z}_2''^{(i-1)}), \ldots, \pi(\mathbf{z}_{n_i}''^{(i-1)})\right) \in S_j^{i-1}, \forall j \in [n_{i-1}],$$

which means that $\lambda\mathbf{z}^{(i-1)} + (1-\lambda)\mathbf{z}'^{(i-1)} \in P^{i-1}$*. Then* P^{i-1} *is convex. By iteratively applying the above two steps backward, it can be shown that there exists* $S^0 \subset \mathbb{R}^n$ *that satisfies the conditions for neurons in layer* i*. The intersection of all those layers is also convex, which means that the activation region for the pattern* $\mathcal{A}$ *is convex.*

Lemma 3 suggests that, given a monotonic activation function π and an activation pattern $\mathcal{A}$, a convex region $\mathcal{R}$ is uniquely determined by $\mathcal{A}$. In other words, the mapping from activation region $\mathcal{R}$ to $\mathcal{A}$ is injective. As most activation functions have distinguishable activated and deactivated regions, the separation on π provides insights into the neuron-level reaction to the input.

4.2.5 Continuity

With convexity, given an activation region $\mathcal{R}$, for any $\mathbf{x}, \mathbf{x}' \in \mathcal{R}$, the segment connecting $\mathbf{x}$ and $\mathbf{x}'$ is completely included in the region. Furthermore, if all the activation regions are convex, the straight line connecting any two points in $\mathbb{R}^n$ can only cross each activation region once. This property is useful in building the

connection between model performance and the similarity of activation patterns, which will be discussed in the next section.

Another useful property of the mapping function is the continuity of the local mapping function. In fact, as long as the breakpoints of the activation function are included in the separation Γ, the mapping within each activation region is continuous.

Lemma 4 (Continuity)**.** *Let $\mathcal{N}$ be a feedforward neural network as defined in Section 4.1 with a monotonic activation function π. Given a continuous separation Γ of π, the mapping from $\mathcal{R}(\mathcal{A};\theta,\sigma,\Gamma) \to f(\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma))$ is continuous.*

Proof (Lemma 4)**.** *The proof of Lemma 4 is intuitive. Since Γ is a continuous operation on the network $\mathcal{N}$ with activation π, h_i is a continuous function. For $i = 1$, $\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma)$ is compact, therefore $h_i(\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma))$ is also compact. For any $i \in 2,\ldots,n$, $h_d \circ h_{d-1} \circ \cdots \circ h_i$ is a composition of continuous functions, and is therefore continuous. Thus, the local mapping function of the network $\mathcal{N} : \mathcal{R}(\mathcal{A};\theta,\pi,\Gamma) \to f(\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma))$ is continuous.*

4.2.6 Lipschitz Bound

The continuity of the activation region enables the analysis of the Lipschitz property of an activation region. Notice that given a neuron (i,j), its pattern $a_j^{(i)}$, and the activation function π, the post-activation of its output with respect to the input vector is bounded. Formally:

Lemma 5 (Layer Lipschitz Bound)**.** *Denote $\mathcal{R}(\mathcal{A};\theta,\pi,\Gamma)$ as an activation region defined on network $\mathcal{N}$. Given $\mathbf{x},\mathbf{x}' \in \mathcal{R}(\mathcal{A};\theta,\pi,\Gamma)$, the input and output the i-th layer is denoted as $x^{(i)}(\mathbf{x})$there is a Lipschitz constant ρ_i for layer i that:*

$$\delta(x^{(i)}(\mathbf{x}), x^{(i)}(\mathbf{x}')) \leq D(\mathbf{x},\mathbf{x}'), \tag{4.17}$$

where $\delta(\mathbf{x},\mathbf{x}') = \|f(\mathbf{x}) - f(\mathbf{x}')\|_2$ and $D(\mathbf{x},\mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2$. The Lipschitz constant

ρ_i is:

$$\rho_i = \max_j \sup_{x \in U_{\hat{a}_j^{(i)}}} \frac{\partial \pi(x)}{\partial x} \cdot Norm(W^{(i)}), \tag{4.18}$$

where $\hat{a}_j^{(i)}$ is the activation pattern of neuron (i, j) and $U_{\hat{a}_j^{(i)}}$ is corresponding interval $(\gamma_{\hat{a}_j^{(i)}-1}, \gamma_{\hat{a}_j^{(i)}})$, and $Norm(W^{(i)})$ is the spectral norm of weights in layer i.

Proof (Lemma 5)**.** *Lemma 5 can directly be obtained from the nature of activation function. A neural network layer can be viewed as composition of functions:*

$$y_j^{(i)}(\mathbf{x}) = \pi(\mathbf{z}_j^{(i)}(\mathbf{x})) \tag{4.19}$$

where $y_j^{(i)}(\mathbf{x}), \mathbf{z}_j^{(i)}$ are the j-th element of output and pre-activation at layer i. Given an activation pattern for all the neurons at layer i, the non-linearity provided by the layer is then the maximum slope of the corresponding domain:

$$D(y^{(i)}(\mathbf{x}), y^{(i)}(\mathbf{x})) \leq \rho_\pi D(\mathbf{z}^{(i)}(\mathbf{x}), \mathbf{z}^{(i)}(\mathbf{x}')), \tag{4.20}$$

where ρ_π is defined as:

$$\rho_\pi = \max_j \sup_{x \in U_{\hat{a}_j^{(i)}}} \frac{\partial \sigma(x)}{\partial x}. \tag{4.21}$$

At the same time, the mapping from input $x^{(i)}$ to pre-activation $\mathbf{z}^{(i)}$ is linear transformation as per defined by Equation 4.4, of which the Lipschitz constant is the norm of equivalent matrix $Norm(W^{(i)})$.

$$D(\mathbf{z}^{(i)}(\mathbf{x}), \mathbf{z}^{(i)}(\mathbf{x}')) \leq \rho_{\psi \circ \phi} D(x^{(i)}(\mathbf{x}), x^{(i)}(\mathbf{x}')), \tag{4.22}$$

where

$$\rho_{\psi \circ \phi} = Norm(W^{(i)}) \tag{4.23}$$

With the composition of functions, the norm of layer i can be written as:

$$\begin{aligned} \rho_i &= \rho_\pi \times \rho_{\psi \circ \phi} \\ &= \max_j \sup_{x \in U_{\hat{a}_j^{(i)}}} \frac{\partial \pi(x)}{\partial x} \cdot Norm(W^{(i)}). \end{aligned} \tag{4.24}$$

By generalizing the above Lemma to the network, we then yield the Lipschitz continuity of the network $\mathcal{N}$ within an activation region.

Lemma 6. *Given an activation region $\mathcal{R}(\mathcal{A}; \theta, \pi, \Gamma)$ where Γ is a continuous separation of net $\mathcal{N}$ with depth n, then for any $x_1, x_2 \in R$, there exist an upper Lipschitz constant ρ that in region $\mathcal{R}(\mathcal{A})$:*

$$\delta(\mathbf{x}, \mathbf{x}') \leq \rho D(\mathbf{x}, \mathbf{x}'), \tag{4.25}$$

where $D(\mathbf{x}, \mathbf{x}')$ is the distance between $\mathbf{x}$ and $\mathbf{x}'$, $\delta(\mathbf{x}, \mathbf{x}')$ is the distance between prediction $f(\mathbf{x})$ and $f(\mathbf{x}')$. The Lipschitz constant can be represented as:

$$\rho = \prod_{i}^{n} \rho_i, \tag{4.26}$$

where ρ_i is the Lipschitz constant of layer i defined in Equation 4.18.

Proof (Lemma 6)**.** *Lemma 6 can directly obtain by using the Lipschitz property of composition of functions. Given a function:*

$$f = f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ \sigma \circ T_1,$$

The Lipschitz constant of f is :

$$\rho_f = \prod_{i}^{n} \rho_i, \tag{4.27}$$

where ρ_i is the Lipschitz constant of its components.

Empirically, the ρ can also be estimated by [254]:

$$\rho = \sup_{\epsilon, \mathbf{x} \in R,} \frac{\| \bigtriangledown f(x) \epsilon \|_2}{\|\epsilon\|_2} = \sup_{\mathbf{x} \in R} \|J_f(x)\|_2, \tag{4.28}$$

$\|J_f(x)\|_2$ is the Jacobian matrix of function f at $\mathbf{x}$.

4.3 From Single Region to Its Neighbor

Now that a general picture of activation regions in the input domain is presented, the investigation can be extended to a larger scope with a focus on describing

the connections between activation regions. Non-linearity is one of the biggest obstacles in understanding deep neural networks. With the activation pattern, it is possible to gain insight into the behavior of the network $\mathcal{N}$ locally. However, as the complexity of $\mathcal{N}$ grows, the average volume of an activation region decreases [170]. This means that studying the properties of a single activation region can only provide limited insights into the behavior of a neural network. To address this gap, this work introduces definitions to describe subspaces $\mathcal{R}$ within the input space by covering it with a union of multiple activation regions. This broadens the scope of the investigation and allows for the study of regional properties of complex neural networks.

In this section, the relationship between an activation region and its neighbors is first described and investigated. It is shown that close regions are distinguished by certain neurons and can be merged into a larger region by relaxing the constraints on those neurons. Finally, the properties of the neural network on a larger scale are discussed, with the objective of investigating the expressive ability and robustness of the neural network on a broader scope.

4.3.1 Adjacent Activation Regions

This section discusses the distance between the predictions of data in different activation regions. Given a neural network $\mathcal{N}$ with activation π, the continuous separation Γ splits the input space into many connected components. Formally, if $\boldsymbol{\theta}$ and Γ are measure-zero sets with respect to the Lebesgue measure, the separation is given by hyperplane arrangements in $\mathbb{R}^{in}$ as defined by Equation 4.11.

Intuitively, given i, j, and k, there exist pairs of activation regions $(\mathcal{R}_1, \mathcal{R}_2)$ that are partitioned by the bent hyperplane H_{ijk}. In other words, $\mathcal{R}_1$ and $\mathcal{R}_2$ are adjacent activation regions. Since they are separated by H_{ijk}, the activation patterns for $\mathcal{R}_1$ and $\mathcal{R}_2$ are identical except for the pattern of neuron (i, j), on which the hyperplane is defined. The following definition is introduced to describe the adjacency of activation regions:

Definition 2 (Adjacent Activation Regions)**.** *Denote $\mathcal{N}$ is a deep network with activation σ and parameters $\boldsymbol{\theta}$. Γ is a continuous separation of N. $\mathcal{R}_1$ and $\mathcal{R}_2$ are two activation regions, $H_{ijk}(\boldsymbol{\theta})$ is a hyperplane defined by neuron (i,j) and breakpoint γ_k. For any $\mathbf{x}_1 \in \mathcal{R}_1$ and $\mathbf{x}_2 \in \mathcal{R}_2$, if:*

$$h_{pqr}(\mathbf{x}_1, \mathbf{x}_2) \begin{cases} <0, p = i, q = j, r = k; \\ >0, \textit{otherwise}, \end{cases} \tag{4.29}$$

where $h_{pqr}(\mathbf{x}_1, \mathbf{x}_2) = (z_q^{(p)}(\mathbf{x}_1; \boldsymbol{\theta}) - \gamma_r) \times (z_q^{(p)}(\mathbf{x}_2; \boldsymbol{\theta}) - \gamma_r)$, then $\mathcal{R}_1$ and $\mathcal{R}_2$ are adjacent activation regions separated by hyperplane $H_{ijk}(\boldsymbol{\theta})$.

The $h_{pqr}(\mathbf{x}_1, \mathbf{x}_2)$ above is introduced to indicate whether two data points $\mathbf{x}_1$ and $\mathbf{x}_2$ are on the same side of a bent-hyperplane. Given neuron index (p, q) and data points $\mathbf{x}_1$ and $\mathbf{x}_2$, if $z^{(p)}q(\mathbf{x}_1; \boldsymbol{\theta}) - \gamma_r$ and $z^{(p)}q(\mathbf{x}_2; \boldsymbol{\theta}) - \gamma_r$ have different signs, then they are on different sides of a bent-hyperplane of $Hpqr(\boldsymbol{\theta})$, and vice versa. If $\mathbf{x}_1$ and $\mathbf{x}_2$ are on the same side of all hyperplanes except for $Hpqr(\boldsymbol{\theta})$, then $\mathbf{x}_1$ and $\mathbf{x}_2$ are in two adjacent activation regions that are separated by H_{pqr}. Geometrically, two adjacent activation regions can be merged into one by removing the hyperplane that separates them. Therefore, every $\mathbf{x}_1 \in \mathcal{R}_1$ and $\mathbf{x}_2 \in \mathcal{R}_2$ are on the same side of all other hyperplanes except for the one that separates them.

Consider two data points, $\mathbf{x}$ and $\mathbf{x}'$, in two adjacent activation regions divided by the hyperplane H_{ijk}. The following lemma shows that two adjacent regions can be uniquely differentiated by a neuron:

Lemma 7. *Assume that $\mathbf{x}$ and $\mathbf{x}'$ locate within adjacent activation regions. There exists a unique neuron(i,j) such that the activation pattern of (i,j) for $\mathbf{x}$ and $\mathbf{x}'$ are different:*

$$\exists!(i,j), \hat{a}_j^{(i)}(\mathbf{x}) \neq \hat{a}_j^{(i)}(\mathbf{x}'). \tag{4.30}$$

Proof (Lemma 7)**.** *Given $\mathbf{x} \in \mathcal{R}, \mathbf{x}' \in \mathcal{R}'$, where $\mathcal{R}$ and $\mathcal{R}'$ are adjacent activation regions that separated by H_{ijk}. Then*

$$h_{pqr}(\mathbf{x}_1) \cdot h_{pqr}(\mathbf{x}_2) > 0, \forall p \neq i, q \neq j. \tag{4.31}$$

Given $p \neq i, q \neq j$, without loss of generality, assume that the pattern of neuron (p, q) is γ_{pq} for $\mathbf{x}$. Then

$$z_q^{(p)}(\mathbf{x}; \boldsymbol{\theta}) - \gamma_r > 0, r = 0, 1, \ldots, r_{pq}$$

Integrating above equation with equation 4.31, $\mathbf{x}'$ satisfies:

$$z_q^{(p)}(\mathbf{x}'; \boldsymbol{\theta}) - \gamma_r > 0, r = 0, 1, \ldots, r_{pq}.$$

This means that the pattern of neuron (p, q) is also r_{pq} for $\mathbf{x}'$. Similarly, it can be derived that the pattern of neuron (i, j) for $\mathbf{x}$ and $\mathbf{x}'$ are k and $k - 1$. This means that (i, j) is the unique neuron that has different pattern for $\mathbf{x} \in \mathcal{R}$ and $\mathbf{x}' \in R'$.

The above lemma shows that, given adjacent activation regions $\mathcal{R}$ and $\mathcal{R}'$, all the other neurons have the same activation status in both regions $\mathcal{R}$ and $\mathcal{R}'$ except for neuron (i, j), on which the hyperplane separating $\mathcal{R}$ and $\mathcal{R}'$ is defined. By removing the bent hyperplane, activation regions $\mathcal{R}$ and $\mathcal{R}'$ can be merged into one region. Moreover, the difference between the mappings in $\mathcal{R}$ and $\mathcal{R}'$ originates from neuron (i, j). This means that regions $\mathcal{R}$ and $\mathcal{R}'$ share common properties that can be described by a set of neurons.

4.3.2 Incomplete Activation Region

This section generalizes the above discussion about adjacent activation regions. Similar to the case of two regions, a set of activation regions can be merged into one by removing a set of bent hyperplanes. The merged region can also be described by a subset of neurons, which is referred to as an incomplete activation pattern as defined below.

Definition 3 (Incomplete Pattern / Region)**.** *Let $\mathcal{N}$ be a neural network defined as in Section 4.1. Denote Γ as a continuous separation of the network. Given an activation pattern $\mathcal{A}$ and a subset of the index set $\mathcal{I}^c \in \mathcal{I}$, denote $\mathcal{A}_{\mathcal{I}^c} \subset \mathcal{A}$ is an incomplete activation pattern of A:*

$$\mathcal{A}_{\mathcal{I}^c} := \{a_j^{(i)} \ | a_j^{(i)} \in \mathcal{A}, (i, j) \in \mathcal{I}^C\}.$$

The activation region of $\mathcal{A}_{\mathcal{I}^c}$ is denoted as:

$$\mathcal{R}(\mathcal{A}^c) = \bigcap_{(i,j)\in\mathcal{I}^c} \{\mathbf{x} \in \mathbb{R}^n | z_j^{(i)}(\mathbf{x}) \in U_{a_j^{(i)}}\}.$$

To understand the above definition, it is necessary to explain the essence of the definition of an activation pattern. Equation 4.9 shows that the activation pattern $\hat{\mathcal{A}}(\mathbf{x})$ introduces a constraint on the pre-activation value of each neuron. Given neuron (i, j) and its pattern k, a subspace $R \subset \mathbb{R}^n$ is defined based on this constraint. By integrating the constraints on each neuron, an activation region is defined for a pattern. This means that the activation region $\mathcal{R}(\mathcal{A})$ of an activation pattern $\mathcal{A}$ is found by identifying all x that satisfy a certain constraint determined by the activation pattern. Therefore, the activation region can essentially be expressed as the intersection of a set of subspaces defined by each of the neurons:

$$\mathcal{R}(\mathcal{A}) = \bigcap_{\forall i,j} \{x \in \mathbb{R}^n | z_j^{(i)}(x) \in U_{\hat{a}_j^{(i)}}\} \tag{4.32}$$

This reveals that the essence of the operator $\mathcal{R}(\cdot)$ is to find all $\mathbf{x}$ satisfying a certain constraint determined by the activation pattern. An incomplete activation pattern is a subset of an activation pattern. Given a subset of indexes $\mathcal{I}^c \subset \mathcal{I}$, the definition of an incomplete activation region involves releasing the constraints on the neuron set $\mathcal{I}\backslash\mathcal{I}^c$, which means removing the intersection in Equation 4.32. In Section 4.2, it is shown that for each neuron, the breakpoints define a set of bent hyperplanes that separate the larger region into smaller ones. In other words, removing the constraint on a neuron (i, j) is equivalent to merging several of the small activation regions into a larger one.

4.3.3 Float and Fixed Neurons

One limitation of the incomplete activation region is that the merged region is defined by the activation status of neurons. It is challenging to identify such a region due to the high complexity of neural networks. Moreover, such a merged region is more likely to be irregular, as it is surrounded by a set of bent hyperplanes.

In studying the interpretability of neural networks, a regularly shaped subspace is more commonly used and preferred, such as a sphere $B_p(\mathbf{x})$ defined by the p-norm, centered at $\mathbf{x}$ with radius r. To address this issue, the following definition introduces float and fixed neurons within a region.

Definition 4 (Float and Fixed Neuron)**.** *Let $\mathcal{N}$ be a neural network defined as in Section 4.1. Given any subset $\mathcal{R} \subset \mathbb{R}^n$, a neuron (i, j) is said to be a fixed neuron of $\mathcal{R}$ if it has the same pattern for any $\mathbf{x} \in \mathcal{R}$; otherwise, it is a float neuron. The collections of fixed neurons and float neurons are denoted as $\mathcal{I}^X(\mathcal{R})$ and $\mathcal{I}^T(\mathcal{R})$, respectively:*

$$\mathcal{I}^X(\mathcal{R}) = \{(i,j) \mid \exists \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{R}, \hat{a}_j^{(i)}(\mathbf{x}_1) \neq \hat{a}_j^{(i)}(\mathbf{x}_2)\}$$

$$\mathcal{I}^T(\mathcal{R}) = \{(i,j) \mid \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{R}, \hat{a}_j^{(i)}(\mathbf{x}_1) = \hat{a}_j^{(i)}(\mathbf{x}_2)\}$$

where $\hat{a}_j^{(i)}(\mathbf{x}_1)$ is defined in Definition 1.

The objective of introducing float neurons and fixed neurons is to provide an additional way to describe the neural network within the activation pattern framework. In fact, the idea of activation patterns and activation regions is to build a connection between the status of neurons and the corresponding region $\mathcal{R}$ in the input domain. Definitions 3 and 4 serve the same purpose but from a regional perspective instead of focusing on a single region. The difference between the two definitions lies in the starting point of the description of the relationship between region and pattern. In particular,

- The incomplete activation pattern defines a region with a subset $\mathcal{I}^c$ of activation patterns. The subspace $\mathcal{R} \subset \mathbb{R}^n$ is obtained by removing the constraints on the complementary set of neurons $\mathcal{I}/\mathcal{I}^c$ and merging the corresponding activation regions into a larger subspace.

- The fixed neuron/float neuron concept describes the stability of neurons within an arbitrary subspace $\mathcal{R} \subset \mathbb{R}^n$.

For the completeness of the definitions, it is necessary to show the relationship between the above definitions. The following lemma demonstrates that a subspace

$\mathcal{R}$ can be covered by the incomplete activation region defined by the fixed neurons within it.

This provides a more straightforward way to understand the incomplete activation pattern (region). Given a subset of indices $\mathcal{I}^c \subset \mathcal{I}$, releasing the constraints on the post-activations indexed by $\mathcal{I}\backslash\mathcal{I}^c$ is equivalent to removing the related hyperplanes and merging the regions together. This means that for any $\mathbf{x} \in \mathcal{R}(\mathcal{A}_{\mathcal{I}^c})$, $\hat{a}^{(i)}j(\mathbf{x})$ is not fixed, and therefore (i, j) is a float neuron. On the other hand, the activation status for all the neurons $(i, j) \in \mathcal{I}^c$ remains the same for any $\mathbf{x} \in \mathcal{R}(\mathcal{A}\mathcal{I}^c)$.

Lemma 8. *Let $\mathcal{N}$ be a neural network defined as in Section 4.1. Given $\mathcal{R} \subset \mathbb{R}^{n_0}$, denote $\mathcal{I}^X$ and $\mathcal{I}^T$ as the sets of fixed neurons and float neurons in $\mathcal{R}$, respectively. Then:*

1. $\mathcal{I}^X(R) \bigcup \mathcal{I}^T(R) = \mathcal{I}$

2. $R \subset \mathcal{R}(\mathcal{A}_{\mathcal{I}^X})$

Proof (Lemma 8)**.** *Given a neuron (i, j) and a region $\mathcal{R} \subset \mathbb{R}^n$, denote the activation pattern of neuron (i, j) as $\hat{a}ij(\mathbf{x})$ for any $\mathbf{x} \in \mathcal{R}$. Since $z^{(i)}j(\mathbf{x}) \in (-\infty, \infty)$, the pattern of $\mathbf{x}$ is either an index k of the region when $z^{(i)}j(\mathbf{x}) \in U_k$, for $k = 1, 2, \ldots, q$, or -1 when $z^{(i)}j(\mathbf{x})$ is located on the bent hyperplane $z_j^{(i)}(\mathbf{x}) = k$, for $k = 0, 1, \ldots, q$.*

This means every neuron has a pattern for $\mathbf{x} \in \mathcal{R}$. If for every $\mathbf{x} \in \mathcal{R}$, the pattern of neuron (i, j) remains the same, then (i, j) is a fixed neuron. Otherwise, it is a float neuron. In other words, a neuron is either fixed or float in $\mathcal{R}$. This implies that $\mathcal{I}^X(\mathcal{R}) \cup \mathcal{I}^T(\mathcal{R}) = \mathcal{I}$.

Now consider statement 2. For any $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{R}$, and a neuron (i, j), if $\hat{a}ij(\mathbf{x}_1) = \hat{a}ij(\mathbf{x}_2)$, then (i, j) is a fixed neuron in $\mathcal{R}$: $(i, j) \in \mathcal{I}^X(\mathcal{R})$. Denote the pattern as a_{ij}. Then for all $\mathbf{x} \in \mathcal{R}$, $z_{ij;\mathbf{x},\boldsymbol{\theta}} = z_{ij;\mathbf{x}',\boldsymbol{\theta}}$. Therefore, $\mathbf{x}_1 \in \mathcal{R}(\mathcal{A}^X)$.

The first statement in Lemma 8 is trivial. According to Definition 4, a neuron is either fixed or float in a given subspace $\mathcal{R} \in \mathbb{R}^n$. The second statement in Lemma 8 establishes a connection between the incomplete activation pattern and

fixed neurons. It shows that for a given region $\mathcal{R}$ with float neuron set $\mathcal{I}^{\mathcal{T}}$, the corresponding incomplete activation pattern of $\mathcal{I}\backslash\mathcal{I}^{\mathcal{T}}$ covers the region $\mathcal{R}$. Thus, every neuron $(i,j) \in \mathcal{I}\backslash\mathcal{I}^{\mathcal{T}}$ has the same activation status in region $\mathcal{R}$.

Based on previous works, this section introduces incomplete activation patterns and fixed/float neurons to describe the stability of neurons in a subspace $\mathcal{R} \in \mathbb{R}^n$. The incomplete activation region defines a subspace by removing the constraints on a set of neurons and merging a set of activation regions together, while the fixed/float neurons describe the stability of neurons in a subspace.

The proposed definitions generalize the study of neural network interpretability to a larger scope beyond being limited to a single region. Now that the definitions and concepts for describing the neural network are presented, the following section will focus on connecting the mapping function of the neural network with its activation pattern.

4.3.4 Upper Bound of Prediction Difference

This section discusses the upper bound of prediction differences between $\mathbf{x}$ and $\mathbf{x}'$ under the framework of activation patterns. Similar to the previous section, it starts by investigating the case of adjacent activation regions and then generalizes the case to a larger scope. It is shown that given $\mathbf{x}$ and $\mathbf{x}' \in \mathbb{R}^n$, the upper bound of the difference between $f(\mathbf{x})$ and $f(\mathbf{x}')$, denoted as $\delta(\mathbf{x}, \mathbf{x}')$, is larger if $\mathbf{x}$ and $\mathbf{x}'$ are within different activation regions.

Intuitively, if two adjacent activation regions $\mathcal{R}_1$ and $\mathcal{R}_2$ are separated by a bent hyperplane, there exist $\mathbf{x} \in \mathcal{R}_1$ and $\mathbf{x}' \in \mathcal{R}_2$ that are close enough.

Lemma 9. *If $\mathcal{R}_1$ and $\mathcal{R}_2$ are adjacent activation regions separated by H_{ijk}, then for an arbitrary small $\epsilon > 0$, there exist $x_1 \in \mathcal{R}_1$ and $x_2 \in \mathcal{R}_2$ s.t.*

$$D(\mathbf{x}_1, \mathbf{x}_2) < \epsilon \tag{4.33}$$

where $D(\cdot,\cdot)$ is a distance metric defined on input space.

Proof (Lemma 9)**.** *Denote H_{ijk} as the hyperplane that separates $\mathcal{R}_1$ and $\mathcal{R}_2$ and (i, j) is the neuron on which the hyperplane is defined. Given an arbitrary small ϵ, for any $\mathbf{x}_m$ on H_{ijk},*

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{x}_m + c \cdot \frac{\partial z_j^{(i)}}{\partial \mathbf{x}_m}(x)/2 \\ \mathbf{x}_2 &= \mathbf{x}_m - c \cdot \frac{\partial z_j^{(i)}}{\partial \mathbf{x}_m}(x)/2, \end{aligned} \tag{4.34}$$

where c is any coefficient that satisfies:

$$c < \epsilon / \| \frac{\partial z_j^{(i)}}{\partial \mathbf{x}}(\mathbf{x}_m) \|. \tag{4.35}$$

It can be derived that:

$$\begin{aligned} \left(W^{(i)} z^{(i)}(\mathbf{x}_1)^T + \beta^{(i)}\right)_j > 0 \\ \left(W^{(i)} z^{(i)}(\mathbf{x}_2)^T + \beta^{(i)}\right)_j > 0 \end{aligned} \tag{4.36}$$

By definition of adjacent activation regions, for any $\mathbf{x}_1 \in \mathcal{R}_1$ and $\mathbf{x}_2 \in \mathcal{R}_2$:

$$h(x_1, z, \gamma) \cdot h(x_2, z, \gamma) \begin{cases} <0, z = z_s, \gamma = \gamma_s, \\ >0, \textit{otherwise} \end{cases} \tag{4.37}$$

This means that $\mathbf{x}_1 \in \mathcal{R}_1$ and $\mathbf{x}_2 \in R_2$ with $D(\mathbf{x}_1, \mathbf{x}_2) < \epsilon$.

This observation is useful for the analysis of model performance on multi-class classification tasks. The next objective of this section is to build a connection between the activation region and model prediction. In particular, it focuses on describing the bounds of prediction differences between data in the same region and data in different regions.

Let $\mathcal{N}$ be a network with activation π and parameters $\boldsymbol{\theta}$ that computes a function $f : \mathbb{R}^n \to \mathbb{R}^c$. Γ is a continuous separation of $\mathcal{N}$. Given data $\mathbf{x}_1 \in \mathcal{R}_1$, the following part of this section considers the data inside $\mathcal{R}_1$ and outside $\mathcal{R}_1$ at the same distance from $\mathbf{x}_1$. That is, $\mathbf{x}_3 \in \mathcal{R}_1$ and $\mathbf{x}_2 \notin \mathcal{R}_1$ with $D(\mathbf{x}_1, \mathbf{x}_2) = D(\mathbf{x}_1, \mathbf{x}_3)$. Denote $\overline{\mathbf{x}_1\mathbf{x}_2}$ and $\overline{\mathbf{x}_1\mathbf{x}_3}$ as the lines connecting $\mathbf{x}_1$ to $\mathbf{x}_2$ and $\mathbf{x}_1$ to $\mathbf{x}_3$, respectively.

To simplify the analysis here, assume that $\mathbf{x}_2$ is located in an adjacent region of $\mathcal{R}_1$, denoted as $\mathcal{R}_2$. $\mathcal{R}_1$ and $\mathcal{R}2$ are separated by the hyperplane $Hijk(\boldsymbol{\theta})$, and

$\overline{\mathbf{x}_1\mathbf{x}_2}$ does not cross any other activation regions. Notice that this assumption can be removed in the formal statement of the theorem. Denote the piecewise function on each region as $f_i : \mathcal{R}_i \to \mathbb{R}^c$. From Lemma 6, there is a Lipschitz constant ρ_i for each of the f_i. From Lemma 9, given an arbitrarily small $\epsilon > 0$, there exist $m_1 \in \mathcal{R}_1$ and $m_2 \in \mathcal{R}_2$ on $\overline{\mathbf{x}_1\mathbf{x}_2}$ such that:

$$\begin{aligned} D(m_1, H_{12}(\gamma, \boldsymbol{\theta})) < \epsilon, \\ D(m_2, H_{12}(\gamma, \boldsymbol{\theta})) < \epsilon. \end{aligned} \tag{4.38}$$

where $D(m_i, H_{ijk}(\gamma, \boldsymbol{\theta}))$ is distance between m_i to the hyperplane $H_{ijk}(\gamma, \boldsymbol{\theta})$.

According to Lemma 6 and triangle inequality of norms, when $\epsilon \to 0$ we have:

$$\begin{aligned} \delta(\mathbf{x}_1, \mathbf{x}_2) \leq \rho_1 D(\mathbf{x}_1, m_1) + \rho_2 D(\mathbf{x}_2, m_2) \\ \delta(\mathbf{x}_1, \mathbf{x}_3) \leq \rho_1 D(\mathbf{x}_1, \mathbf{x}_3) \end{aligned} \tag{4.39}$$

For a network with high complexity, almost surely there exists an adjacent activation region of $\mathcal{R}_1$, denoted as $\mathcal{R}_h$, that satisfies $\rho_h > \rho_1$, where ρ_h is the Lipschitz constant of activation region $\mathcal{R}_h$. Therefore, the upper bound of $\delta(\mathbf{x}_1, \mathbf{x}_3)$ is lower than that of $\delta(\mathbf{x}_1, \mathbf{x}_2)$. By nature, the mapping of network $\mathcal{N}$ is equivalent to the combination of all the piecewise functions. The above equation can be generalized to the network:

Theorem 2. *Let $\mathcal{N}$ be a neural network with mapping function $f : \mathbb{R}^n \to \mathbb{R}^c$, parameters $\boldsymbol{\theta}$, and activation π. Γ is a continuous separation of $\mathcal{N}$. For any $\mathbf{x} \in \mathbb{R}^n$, denote the activation region in which $\mathbf{x}$ is located as $\mathcal{R}$. Given $r > 0$, almost surely:*

$$\sup_{\substack{\mathbf{x}' \in \mathcal{R}, \\ D(x,x')=r}} \delta(\mathbf{x}, \mathbf{x}') \leq \sup_{\substack{\mathbf{x}' \notin \mathcal{R}, \\ D(x,x')=r}} \delta(\mathbf{x}, \mathbf{x}'). \tag{4.40}$$

Proof. *Given data $\mathbf{x}_1 \in \mathcal{R}_1$, assume that $\mathbf{x}_3 \in \mathcal{R}_1$ and $\mathbf{x}_2 \notin \mathcal{R}_1$ with $D(\mathbf{x}_1, \mathbf{x}_2) = D(\mathbf{x}_1, \mathbf{x}_3)$. Denote the activation regions cross by line $\overline{\mathbf{x}_1\mathbf{x}_2}$ are $\mathcal{R}_2, \mathcal{R}_3, \cdots, \mathcal{R}_n$ with Lipschitz constant $\rho_2, \rho_3, \cdots, \rho_n$. Denote $c_i (i = 1, \ldots, n-1)$ as the intersections between $\overline{\mathbf{x}_1\mathbf{x}_2}$ and hyperplane that separates R_i, R_{i+1}. From Lemma 9, for any $\epsilon > 0$, a sequence of points $s_i \in R_i$ and $e_i \in R_i$ can be found on $\overline{\mathbf{x}_1\mathbf{x}_2}$ such that:*

$$\begin{aligned} D(c_i, s_{i-1}) < \epsilon \\ D(e_i, c_i) < \epsilon. \end{aligned} \tag{4.41}$$

In other words, s_i and e_i can be viewed as start point and end point on segment in region R_i.

According to Lemma 6 and triangle inequality of norms, when $\epsilon \to 0$ we have:

$$\begin{aligned}\delta(\mathbf{x}_1, \mathbf{x}_2) &\leq \rho_1 D(\mathbf{x}_1, e_1), + \sum_{i=2}^{n-1} \rho_i D(s_i, e_i) + \rho_n D(s_n, \mathbf{x}_2) \\ \delta(\mathbf{x}_1, \mathbf{x}_3) &\leq \rho_1 D(\mathbf{x}_1, \mathbf{x}_3)\end{aligned} \tag{4.42}$$

Consider the simple case where $n = 2$. For any activation region $\mathcal{R}$, with probability 1 there exist adjacent activation region R' that $\rho < \rho'$, for any $\mathbf{x}_2 \in R'$,

$$\rho_1 D(\mathbf{x}_1, e_1) + \rho_2 D(s_2, \mathbf{x}_2) \geq \rho_1 D(\mathbf{x}_1, \mathbf{x}_3).$$

Then,

$$\sup_{\substack{x' \in \mathcal{R}, \\ D(x,x')=r}} \delta(\mathbf{x},) \|F(x) - F(x')\|_2 \leq \sup_{\substack{x' \in \mathcal{R}', \\ D(x,x')=r}} \|F(x) - F(x')\|_2. \tag{4.43}$$

For $n > 2$, since the volume of each region are bounded, with the constraint that $D(\mathbf{x}_1, \mathbf{x}_2) = D(\mathbf{x}_1, \mathbf{x}_3)$, the average number of n is limited. We can iteratively find a sequence of regions R_i with $\rho_i > \rho_0$. Therefore,

$$\sup_{\substack{\mathbf{x}' \in \mathcal{R}, \\ D(x,x')=r}} \delta(\mathbf{x}, \mathbf{x}') \leq \sup_{\substack{\mathbf{x}' \notin \mathcal{R}, \\ D(x,x')=r}} \delta(\mathbf{x}, \mathbf{x}'). \tag{4.44}$$

In other words, for network $\mathcal{N}$, the bounds of the spread between the predictions of data in the same activation region are lower than those for data in different activation regions. Similar results can also be obtained for the lower bound. When the number of activation regions is large enough, the volume of each region is bounded [19], and the above theorem can be generalized globally.

Intuitively, for data $\mathbf{x}$ and $\mathbf{x}'$, if there exists a continuous separation and activation pattern $\mathcal{A}$ such that both $\mathbf{x}, \mathbf{x}' \in \mathcal{R}(\mathcal{A}; \boldsymbol{\theta}, \pi, \Gamma)$, then the differences between the predicted results of $\mathbf{x}$ and $\mathbf{x}'$ have a tighter bound. On the other hand, for any two data points $\mathbf{x}$ and $\mathbf{x}'$, if they are located in activation regions that are far apart, then the bound of the difference between their predictions is much looser. Therefore, the similarity between activation patterns can be used as a measure of the topological distance between data on the network.

Figure 4.3: An illustration of generalized activation region for ReLU net with 3 layers, with 64 neurons within each layer. The continuous separation is set as $\Gamma = \{0\}$.

4.4 Geometric Illustration

This section aims to further explain the concepts of activation regions, activation patterns, and activation paths. The first part of this section presents the activation regions of a simple neural network with different activation functions. The second part provides a geometric illustration of the proposed concepts by showing how a set of hyperplanes partitions the input domain into activation regions. Moreover, a subspace $\mathcal{R}$ is demonstrated in the figure to show how the fixed/float neurons and fixed/float paths are defined.

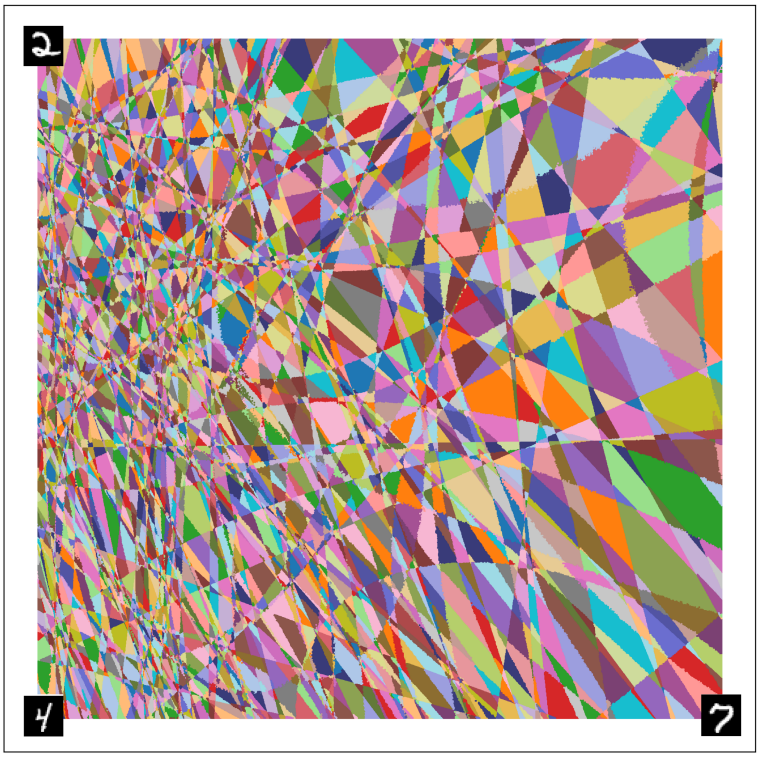


Figure 4.4: An illustration of generalized activation region for Sigmoid net with 3 layers, with 64 neurons within each layer. The continuous separation is set as $\Gamma = \{-1, 1\}$.

4.4.1 Regions

The generalized activation pattern is an extension of linear activation regions. For instance, given ReLU activation and breakpoints $\Gamma = 0$, it is equivalent to the definition from previous works. On the other hand, for continuous activation functions, it provides a way to separate the input domain into different subspaces where each subspace has a different reaction to the input. The generalized activation pattern can be viewed as a toolbox to further investigate model performance with different activations.

To demonstrate the activation region with different activation functions, a simple neural network was constructed and trained on the MNIST dataset to convergence. The MNIST dataset is a collection of 70,000 grayscale images of handwritten digits

from 0 to 9, divided into 60,000 training and 10,000 test samples. Each image is 28x28 pixels, and the dataset is widely used as a benchmark for evaluating machine learning algorithms on image recognition tasks.

After reshaping the 28x28 grayscale image, the input of a data become 784. Therefore, we use a network with an input size of 784, 3 layers with 64 neurons each, and an output size of 10 that corresponding to the 10 digits classes. After training the network on training data, we took 3 test samples and project those 3 data to 2 dimensional space where the data points locate at (0, 0), (1, 0) and (0, 1). Using the same projection function, we project data close to the 3 test samples to the same 2D space and categorize those data into different activation region according to the pre-activation value of neurons. This gives us Figure 4.3 and Figure 4.4, which demonstrate the activation regions of the network with ReLU activation and Sigmoid activation, respectively. It can be observed that, in the 2D projection, the regions in both figures are convex.

Moreover, within the triangle formed by coordinates (0, 0), (1, 0), and (0, 1), the average volume of each region is relatively smaller compared with that outside of it. As the data are projected onto these vertices, the observation implies that on the support of the data distribution $\mathbb{D}_{\mathbf{x}}$, the neural network exhibits higher expressive ability. Compared with the ReLU network, the boundary of the Sigmoid network is curved. Although the change in input is linear, it is bent by the Sigmoid function after the first layer, resulting in a non-linear bent hyperplane.

It is interesting to visually present the activation regions for a neural network. However, the activation regions themselves provide limited information since they are merely a separation of the input space according to the model. To make this concept useful for our analysis, it is necessary to investigate the properties of neural networks using the concepts proposed in this dissertation.

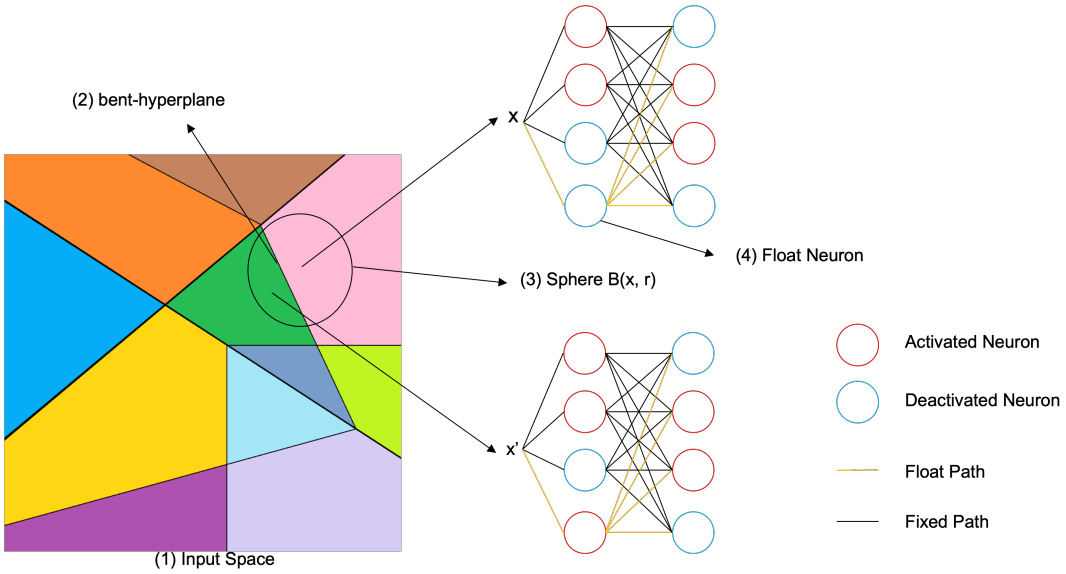


Figure 4.5: An illustration of the fixed (float) path and neuron of a neural network with 2D input, 4D output, and 1 hidden layer. (1) The 2D input space; (2) A bent-hyperplane defined by $H = \{x | z_4^{(1)}(x) = 0\}$; (3) A sphere centered at x; (4) A float neuron with index (1,4) in the network.

4.4.2 Illustration of the Concepts

To help with understanding the proposed concepts, Figure 4.5 presents an illustration of the key concepts in this work. Consider $\mathcal{N}$ as a neural network with a 2D input, 4D output, and 1 hidden layer with 4 neurons. Assume that $\mathcal{N}$ has a ReLU activation function. The input space is partitioned into several regions by a set of hyperplanes. Each region is referred to as an activation region, marked in a unique color, within which the mapping function is linear.

For example, the pink and green regions are divided by the bent hyperplane $H = x \mid z_4^{(1)}(x) = 0$, where $z_4^{(1)}(x)$ is the pre-activation value of the 4th neuron in the 1st layer. Let $B(x, r)$ be a sphere centered at x with radius r. $B(x, r)$ is covered by the union of the pink and green regions. Since the two regions can be merged into one by removing the hyperplane, the incomplete activation pattern of the merged region is $\mathcal{I}\backslash(1, 4)$. Therefore, $(1, 4)$ is the only float neuron in $B(x, r)$. This means

that for every $x \in B(x, r)$, all the neurons have the same activation pattern except for neuron $(1, 4)$.

The illustration also presents the computational paths for $x \in B(x, r)$. Given a path ζ, if neuron $(1, 4)$ is not on this path, then ζ is a fixed path, represented by the black lines in the figure. This means that for every $x \in B(x, r)$, the values of those fixed paths are linear functions with respect to x. On the other hand, all the non-linearity of the function $f(x)$ for $x \in B(x, r)$ is contributed by the float paths (orange lines).

4.5 Summary of the Chapter

This chapter introduces the analytic framework of the research presented in this dissertation, focusing on its connections with previous works and the motivation behind the innovations. As a preliminary to the in-depth analysis in the following chapters, this chapter also briefly discusses the properties of the proposed concepts and how they contribute to understanding neural networks.

Following the common process of research in interpretability, this dissertation begins with a theoretical analysis and then extends the results into several applications. To avoid conflicts and provide a guideline for this work, Section **??** first introduces the notations used in this work, with their meanings summarized. In particular, several tables are presented separately for different parts of this work.

Section 4.2 and Section 4.3 introduce the generalized activation pattern and extend the framework into larger subspaces beyond a single activation region. In particular, it is demonstrated that there exists a larger upper bound for $\mathbf{x}$ and $\mathbf{x}'$ from different activation regions. Based on the discussion in this section, Chapter 5 introduces a *pattern similarity* metric for comparing neural networks with different activations and a *neuron entropy* metric for evaluating the expressive ability of neural networks and identifying the *dying neuron issue*.

To provide a quantified analysis of how inputs affect the output of a neural

network, Section 6.1 introduces the definitions of *fixed path* and *float path* in a subspace $\mathcal{R}$. It is further shown in Chapter 6, where the robustness of models is studied, that float paths contribute significantly to the instability of neural networks by decomposing the computational graph of neural networks with piecewise linear activation functions into fixed and float parts.

Chapter 5

Model Expressivity and Dying Neuron

Chapter 3 demonstrates that the gradient vanishing issue can be effectively addressed through the use of batch normalization and weight initialization techniques. By investigating the training dynamics of neural networks, it is shown that for symmetric activation functions, weight updates are highly dependent on the current weights, which can result in convergence failures. This leads to a *dying neuron issue*, where most neurons fire similar signals regardless of the inputs.

To further understand these observations, this chapter investigates the expressive ability of neural networks, focusing on evaluating the consistency of neuron activation patterns. Specifically, this chapter is divided into two main parts:

- The first part presents a comparison of performance across models with different activation functions. It aims to address the introductory question and explain the performance gap among neural networks with different activation functions. Section 5.1 introduces a *pattern similarity* metric and illustrates its basic properties, connecting it with previously proposed metrics for evaluating model expressive ability. Section 5.2 presents experimental results on pattern similarity and explains the poorer performance of neural networks with saturated activation functions by showing a more severe dying neuron issue.

- The second part focuses on gaining insights using the framework proposed in this dissertation to evaluate the expressive ability of individual models. Section 5.3 introduces a metric named *neuron entropy* that indicates the *uncertainty* of neuron activation patterns. It then demonstrates the connection between this metric and model performance, as well as metrics proposed by previous works. It is found that, in deep-intermediate layers, a large proportion of neurons fire the same signals regardless of the input, leading to a loss of expressive ability. This indicates that some neurons fail to contribute to model performance. Based on this result, Section 5.4 introduces a pruning algorithm to remove redundant neurons.

5.1 Pattern Similarity and Model Performance

This section continues the analysis of neural network properties given continuous separation and aims to utilize the activation pattern to evaluate model expressive ability. In Section 4.3, it is shown that for any two points $\mathbf{x}$ and $\mathbf{x}'$, if there exists a separation provided by Γ and an activation pattern $\mathcal{R}(\pi, \mathcal{A}, \theta, \Gamma)$ such that $\mathbf{x}, \mathbf{x}' \in \mathcal{R}$, an upper bound of $\delta(\mathbf{x}, \mathbf{x}') = |f(\mathbf{x}) - f(\mathbf{x}')|_2$ can be constructed. Moreover, given $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$, the upper bound of $\delta(\mathbf{x}, \mathbf{x}')$ is lower if the data are located within closer activation regions.

Since activation regions are defined by activation patterns, two closer activation regions have similar activation patterns. The intuition here implies that the upper bound $\delta(\mathbf{x}, \mathbf{x}')$ is connected to the difference between the activation patterns of $\mathbf{x}$ and $\mathbf{x}'$. Based on this, a *pattern similarity* metric is introduced to evaluate neural networks based on the neuron-level response to the data.

5.1.1 Pattern Similarity

The objective of this section is to introduce the pattern similarity and establish the connection between the proposed metric and model performance. Despite the

intuitive insights discussed in the previous section, neural networks are complex, and even similar data with the same label can be located within activation regions that are far apart.

To illustrate this connection, the analysis in this section adopts transition density as a bridge, a metric proposed in previous works for evaluating the model expressive ability of neural networks. Using transition density as a bridge, this section shows that for a dataset with a sufficiently large capacity, a high transition density of trajectories connecting pairs of data implies high prediction differences. In fact, a high pattern similarity between $\mathbf{x}$ and $\mathbf{x}'$ means that the two samples are in close activation regions; therefore, $\overline{\mathbf{x}\mathbf{x}'}$ has lower transition density. This means that the upper bound of $\delta(\mathbf{x}, \mathbf{x}')$ is lower.

Transition density of trajectories was proposed in earlier work [182] to measure how many activation regions a segment crosses. Given two close points $\mathbf{x}_1$ and $\mathbf{x}_2$ in the input domain, if the activation patterns are different for $\mathbf{x}_1$ and $\mathbf{x}_2$, a transition is said to have occurred between $\mathbf{x}_1$ and $\mathbf{x}_2$. For a one-dimensional trajectory connecting $\mathbf{x}_0$ and $\mathbf{x}_k$ in the input space, if we sample $k-1$ equidistant points on the trajectory, the transition density is defined as the number of transitions for the set $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$:

$$\mathbf{TD}(\overline{\mathbf{x}_1\mathbf{x}_2}) = \lim_{k\to\text{inf}} \sum_{i=0}^{k-1} \mathbf{Tra}(x_i, x_{i+1}), \tag{5.1}$$

where

$$\mathbf{Tra}(x_i, x_{i+1}) = \begin{cases} 1 & \mathcal{A}(x_i) \neq \mathcal{A}(x_{i+1}) \\ 0 & \text{elsewise} \end{cases} \tag{5.2}$$

It was introduced to evaluate the sensitivity of a network by measuring the transition density of a trajectory around real data. However, this metric is not suitable for evaluating model performance on a dataset due to the following reasons:

- First, the average volume of each activation region is relatively small for a network with high complexity, which means empirically the transition density of a trajectory can be imprecise.

- Second, computing the transition density of numerous trajectories connecting data pairs is a computationally intensive task.

To mitigate this gap, the *pattern similarity* metric is introduced as below.

Definition 5 (Pattern Similiarity)**.** *Denote N as a feedforward network with non-linearity σ and parameters θ. Γ is a continuous separation of net N. θ and Γ are measure zero set with respect to Lebesgue measure. Denote the activation pattern for any $x \in R^{in}$ as:*

$$A(\mathbf{x}; \sigma, \theta, \Gamma) = \{(z, a_z) | z is\ a\ neuron\ in N\},$$

shortened as $AP(x)$ if other settings are fixed. The pattern similarity between two data $x_1, x_2 \in R^{in}$ is then defined as:

$$\mathbf{PS}(x_1, x_2; \pi, \theta, \Gamma) = \frac{\#(AP(x_1) \bigcap AP(x_2))}{\#number\ of\ neurons\ in N}. \tag{5.3}$$

Additionally, given a dataset distribution X, the pattern similarity of dataset X :

$$PS_E(X) = \mathop{\mathbf{E}}_{x_i, x_j \sim X} [\![PS(x_i, x_j; \sigma, \theta, \Gamma)]\!] \tag{5.4}$$

The pattern similarity distribution of X for a given $\lambda > 0$ is:

$$PS_D(X, \lambda) = \mathbf{P}(PS(x_i, x_j; \sigma, \theta, \Gamma) > \lambda | x_i, x_j \sim X) \tag{5.5}$$

Pattern similarity illustrates how the model responds to a single data point as well as to a dataset.

- Equation 5.3 defines the pattern similarity between two data points. It measures the proportion of neurons in a neural network that have the same activation pattern for $\mathbf{x}$ and $\mathbf{x}'$.

- Equation 5.4 extends the definition from a single pair of data to a dataset. Specifically, it measures the expected pattern similarity for two arbitrary data points $\mathbf{x}$ and $\mathbf{x}'$ sampled from a data distribution $\mathbb{D}$.

- Equation 5.5 provides the cumulative distribution of pattern similarity given a data distribution $\mathbb{D}$. It shows how the pattern similarity is distributed across the selected samples.

The provided information is useful in analyzing model performance. With pattern similarity, the expressive ability of different neural networks can be expressed in terms of the average distance of data with different labels. This is detailed in Section 5.2, where the dataset is divided into subsets based on labels, and the pattern similarity across datasets is measured for different models.

5.1.2 Pattern Similarity and Prediction Difference

Before conducting experiments with the proposed metric, it is important to describe the relationship between pattern similarity and prediction difference. This connection is built using the transition density of trajectories as a bridge. Specifically, it is shown that:

- Pattern similarity is negatively correlated with trajectory density.

- Trajectory density is positively correlated with the upper bound of prediction difference $\delta(\mathbf{x}, \mathbf{x}')$.

Therefore, pattern similarity is negatively related to the upper bound of prediction difference $\delta(\mathbf{x}, \mathbf{x}')$. By interpreting these results for different sub-datasets, it can be inferred that pattern similarity should be higher for subsets with a single label and lower for datasets with mixed labels.

Intuitively, pattern similarity measures the neuron-level response to the input. Given a model, high pattern similarity between data with different labels implies that most neurons in the model fail to distinguish between inputs. In other words, the network suffers from a severe *dying neuron* issue.

5.1.2.1 Pattern Similarity and Trajectory Density

Now that the general idea behind pattern similarity is presented, the following discussion aims to connect pattern similarity with trajectory density. This discussion begins with the properties of activation regions as described in Chapter 4.3.

Lemma 3 suggests that activation regions of a network with a monotonous activation function are convex. Here, we show that, given convexity, the transition density of a straight line can only cross each activation region once.

Lemma 10. *Let N be a network with a monotonous activation σ. Given a continuous separation Γ of network N, denote: $H_z(\theta, \gamma_z) := x \in \mathbb{R}^{in} \mid W_z x + b_z = \gamma_z$ as the hyperplanes defined by neuron z. Then, for any two points $x, y \in \mathbb{R}^{in}$, the line l that connects x and y intersects each $H_z(\sigma, \theta, \gamma_z)$ at most once.*

In the following, we denote the set of regions crossed by line l as $S(l)$. As two adjacent activation regions are separated by a hyperplane defined by the state of a neuron, their activation patterns have limited differences. Lemma 10 suggests that, given a continuous separation under certain conditions, a straight line crosses each region only once, which implies that a lower transition density always comes with a higher pattern similarity. However, for any two points located in regions far apart, it is hard to determine the qualitative relationship between their pattern similarity and the transition density of the line connecting them. In particular, for a network with a complex structure, there exist *closed* patterns whose regions are a measure zero set in the input space. Therefore,

$$\mathbf{TD}(\overline{\mathbf{x}_1\mathbf{x}_2}) + \#(AP(x_1) \bigcap AP(x_2)) = \#\text{neurons in N} \tag{5.6}$$

does not always hold.

To describe the relationship between transition density and pattern similarity formally, we need to add additional constraints.

Theorem 3. *Let N be a net with monotonous non-linearity σ, measure zero parameter set θ and Γ is continuous separation of net N. Given dataset distribution*

X_1 and X_2, almost surely following statements are equivalent:

$$\begin{aligned} \mathop{\mathbf{E}}_{x_i,x_j\sim X_1}[\![\mathbf{TD}(\overline{\mathbf{x}_i\mathbf{x}_j})]\!] \leq \mathop{\mathbf{E}}_{x_i,x_j\sim X_2}[\![\mathbf{TD}(\overline{\mathbf{x}_i\mathbf{x}_j})]\!], \\ PS_E(X_1) \geq PS_E(X_2). \end{aligned} \tag{5.7}$$

Proof. *Recall that the activation regions are split by a set of hyperplanes $H_z(\theta,\gamma)$. Two adjacent activation regions R_1, R_2 can be merged into one by removing the hyperplane that separates them, which is equivalent to removing the constraints on the neurons where the hyperplane is defined. Therefore, for $x_1 \in R_1$ and $x_2 \in R_2$, $\mathbf{TD}(\overline{\mathbf{x}_1\mathbf{x}_2}) = 1$.*

Now we consider the difference between the activation patterns of x_1 and x_2. Naturally, there exists at least one pair of (z,γ) that defines the hyperplane: $H_z(\theta,\gamma) := \{x \in R^{in} | W_z z_i n + b_z = \gamma_z\}$. Denote the segment of the hyperplane that separates R_1 and R_2 as H_{12}. Consider another hyperplane H' defined by z', γ', if it satisfies:

$$W_{z'} z'_{in}(x) + b_{z'} = \gamma'_z, \tag{5.8}$$

for any $x \in H_{12}$, where $z'_{in}(x)$ is the input value of neuron z' given input data x, the activation patterns of x_1 and x_2 are different at neurons z and z'. Therefore, x_1 and x_2 have different patterns at z and z'.

In other words, if there exist n hyperplanes that overlap and separate regions R_1 and R_2, then the pattern difference between x_1 and x_2 is n. However, since the parameter set is a measure zero set, with probability 1, Equation 5.8 does not hold. Therefore, almost surely:

$$\#(AP(x_1) \bigcap AP(x_2)) = \#neurons\ in\ N - \mathbf{TD}(\overline{\mathbf{x}_1\mathbf{x}_2})$$

Therefore, for any x_1, x_2, x_3, $x_4 \in R^{in}$, almost surely following statements are equivalent:

$$\begin{aligned} \mathbf{TD}(\overline{\mathbf{x}_1\mathbf{x}_2}) \leq \mathbf{TD}(\overline{\mathbf{x}_3\mathbf{x}_4}) \\ PS(\mathbf{x}_1,\mathbf{x}_2;\sigma,\theta,\Gamma) \geq PS(\mathbf{x}_3,\mathbf{x}_4;\sigma,\theta,\Gamma). \end{aligned} \tag{5.9}$$

Similar result can be generalized to the dataset.

In particular, for a network with a complex structure, there exist closed patterns whose regions are measure zero sets in the input space. Denote the set of those neurons as $Z(R_1, R_2)$. For a merged region, all the points in that region have the same activation pattern. Therefore, for any $x_1 \in R_1$, $x_2 \in R_2$, we have

$$\#(AP(x_1) \bigcap AP(x_2)) = \#neurons\ in\ N - \#Z(\mathcal{R}_1, \mathcal{R}_2)$$

From Lemma 4, we know that when σ is monotonous, l_i only crosses every region in R_i once. Therefore, R_i is a sequence of adjacent regions and can be merged into one as stated above. Denote:

$$R_m = \bigcup_i R_i\ for\ \ R_i \in \{R_1\}$$

as the union of all activation regions in $\{R_1\}$. Since $S(\overline{\mathbf{x}_1\mathbf{x}_2}) \subset S(l_{x_3,x_4})$, we perform the same operation for $\{R\}_2$ to construct a set $\{R\}_2'$:

$$\{R\}_2' = R_m \bigcup \{R_j | for R_j \notin \{R\}_1\}$$

Since line l_1 are within the region R_m while line l_2 crosses every region in $\{R\}_2'$, we have $PS(x_1, y_1) \leq PS(x_2, y_2)$.

Theorem 3 builds a connection between the pattern similarity metric and trajectory transition density. For any two data in the input space, the lower transition density of the segment connecting them implies higher pattern similarity between the data. The result also holds for dataset distributions. Notice that for the data distribution case, the assumption is satisfied with probability 1 therefore can be removed.

5.1.3 Pattern Similarity and Model Performance

This section aims to build a connection between pattern similarity and model expressive ability as well as generalization.

Given a data point $\mathbf{x} \in \mathbb{R}^{in}$, consider two data points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{in}$ with $D(\mathbf{x}, \mathbf{x}_1) = D(\mathbf{x}, \mathbf{x}_2)$. Let $\overline{\mathbf{x}\mathbf{x}_1}$ and $\overline{\mathbf{x}\mathbf{x}_2}$ be the lines connecting $\mathbf{x}, \mathbf{x}_1$ and $\mathbf{x}, \mathbf{x}_2$,

respectively. The two lines can be viewed as subspaces in $\mathbb{R}^n$. Denote the fixed neurons of $\overline{\mathbf{xx}_1}$ and $\overline{\mathbf{xx}_2}$ as $\mathcal{I}_1^X$ and $\mathcal{I}_2^X$. Assume that $\mathcal{I}_1^X \subseteq \mathcal{I}_2^X$. This means that there are more float neurons in line $\overline{\mathbf{xx}_2}$ than in line $\overline{\mathbf{xx}_1}$. In other words, the line $\overline{\mathbf{xx}_2}$ crosses more activation regions than that of $\overline{\mathbf{xx}_1}$.

Lemma 10 suggests that $\overline{\mathbf{xx}_1}$ and $\overline{\mathbf{xx}_2}$ cross each activation region only once. According to the definition of an adjacent activation region, a set of adjacent activation regions can be merged into one region by removing the hyperplanes that separate them. Moreover, Lemma 8 shows how incomplete activation regions are defined by a set of fixed neurons.

Denote R_0 as the incomplete activation pattern defined by the fixed neuron of $\overline{\mathbf{xx}_1}$:

$$R_0 = \mathcal{R}^{A_{\mathcal{I}^X}} \supseteq \bigcup_{R_i} R_i, \tag{5.10}$$

where R_i are the single activation regions that crossed by $\overline{\mathbf{xx}_1}$:

$$\exists x \in R_i s.t. x \in \overline{\mathbf{xx}_1}. \tag{5.11}$$

Then $\overline{\mathbf{xx}_1}$ is within the region R_0. Denote the Lipschitz constant of R_0 as ρ_0, which equals to:

$$\rho_0 = \max_i \rho_i \tag{5.12}$$

where ρ_i is the Lipschitz constant of $R_i \in S(l_1)$.

Next step is to perform the same analysis for line $\overline{\mathbf{xx}_2}$. Since $S(l_1) \subset S(l_2)$, there are other regions crossed by $\overline{\mathbf{xx}_2}$, which are denoted as $R_1, \ldots, R_n$ with $\mathbf{x}_2 \in R_n$. Now the comparison between $D(\mathbf{x}, \mathbf{x}_1)$ and $D(\mathbf{x}, \mathbf{x}_2)$ can be conducted by Theorem 2:

$$\sup_{x_1} D(F(x), F(x_1)) \leq \sup_{x_2} D(F(x), F(x_2)) \tag{5.13}$$

By loosening the constraints of input we have:

$$\sup_{x_1} R(x, x_1) \leq \sup_{x_1} R(x, x_2) \tag{5.14}$$

where $R(x, y) = D(F(x), F(y))/D(x, y)$ is the distance of prediction to distance between data ratio. Formally, we have the following theorem:

Theorem 4. *Let N be a net with monotonous non-linearity σ, measure zero parameter set θ and a continuous separation Γ. Given x, for any $x_1, x_2 \in R^{in}$, if $S(\overline{\mathbf{x}_1\mathbf{x}_2}) \subset S(l_{x_3,x_4})$, then following statements are equivalent:*

$$\begin{aligned} \mathbf{TD}(\overline{\mathbf{x}\mathbf{x}_1}) &\leq \mathbf{TD}(\overline{\mathbf{x}\mathbf{x}_2}) \\ \sup_{x_1} R(x, x_1) &\leq \sup_{x_2} R(x, x_2) \end{aligned} \tag{5.15}$$

In particular, given dataset distribution X_1 and X_2, if:

$$\mathop{\mathbf{E}}_{x_i,x_j\sim X_1}[\![\mathbf{TD}(l_{x_i,x_j})]\!] \leq \mathop{\mathbf{E}}_{x_i,x_j\sim X_2}[\![\mathbf{TD}(l_{x_i,x_j})]\!], \tag{5.16}$$

then

$$\mathop{\mathbf{E}}_{x_i,x_j\sim X_1}[\![\sup_{x_1} R(x_i, x_j)]\!] \leq \mathop{\mathbf{E}}_{x_i,x_j\sim X_2}[\![\sup_{x_1} R(x_i, x_j)]\!] \tag{5.17}$$

Combining Theorem 3 and Theorem 4, the connection between activation pattern, transition density, and the bound of prediction difference can be obtained. To be specific, given data x and x', lower transition density of line l connecting x and x' indicates a tighter upper bound and lower bound of the distance between $F(x)$ and $F(x')$, as well as a high pattern similarity between x and x'. In practice, when evaluating dataset capacity, if it is large enough, the statement still holds empirically, as shown in Figure 5.2. Experiment details will be discussed in Section 5.2.

For any continuous separation, a well-behaved model should satisfy that the difference between predictions is small for data with the same label but large for data with different labels. To illustrate the statement, imagine a worst case of a deep model. Suppose that there are n classes of data and N different connected components provided by hyperplanes, where $N \gg n$. Given a model with fixed parameters θ, if there exists a continuous separation Γ that satisfies for any sample (x, y), where y is the label of x:

$$x \begin{cases} \in R_1, & y \neq 1 \\ \notin R_1, & \text{elsewise} \end{cases} \tag{5.18}$$

Then for any data x_1, x_2 from class $\{2, 3, \ldots, N\}$, we have:

$$\|F_1(x_1) - F_1(x_2)\|_2 \leq \rho_1 \|x_1 - x_2\|_2, \tag{5.19}$$

where ρ_1 is the Lipschitz constant of R_1.

For a set of data with different labels drawn from region R_i, the above equation constrains the distance of their prediction. In other words, the model fails to distinguish the differences between samples with different labels. Conversely, data with label 1 are distributed within $N - 1$ different regions, therefore having lower pattern similarity and a looser bound on the prediction distance. By such means, we are able to use pattern similarity as a metric to evaluate the model expressivity and generalization towards different datasets.

One of the difficulties encountered during understanding deep learning models is that we hardly know how the neurons act in the black box. The *pattern similarity* fills the gap and can be used for evaluating the model expressivity and generalization. Moreover, it provides a glimpse of the inference mechanism of the model.

5.1.4 Illustration of Pattern Similarity and Transition Density

To illustrate the concept of pattern similarity and transition density, Figure 5.1 presents the input space splitted into several activation regions of by a 2 layers neural network, each layer with 4 neurons.

For inputs x and x', it can be noticed that there is only 1 neuron has different activation status among 8 neurons. The differences between the mapping function for x and x' is causeed by the 4-th neuron locates at layer 1, while all the other neurons are the same for those x and x'. Therefore,the pattern similarity between x and x' is:

$$\mathbf{PS}(\mathbf{x}, \mathbf{x}'; \pi, \theta, \Gamma) = 0.875.$$

The transition density, which is defined as number of different activation regions a segment would cross by connecting two data points, for x and x' is 1 as presented

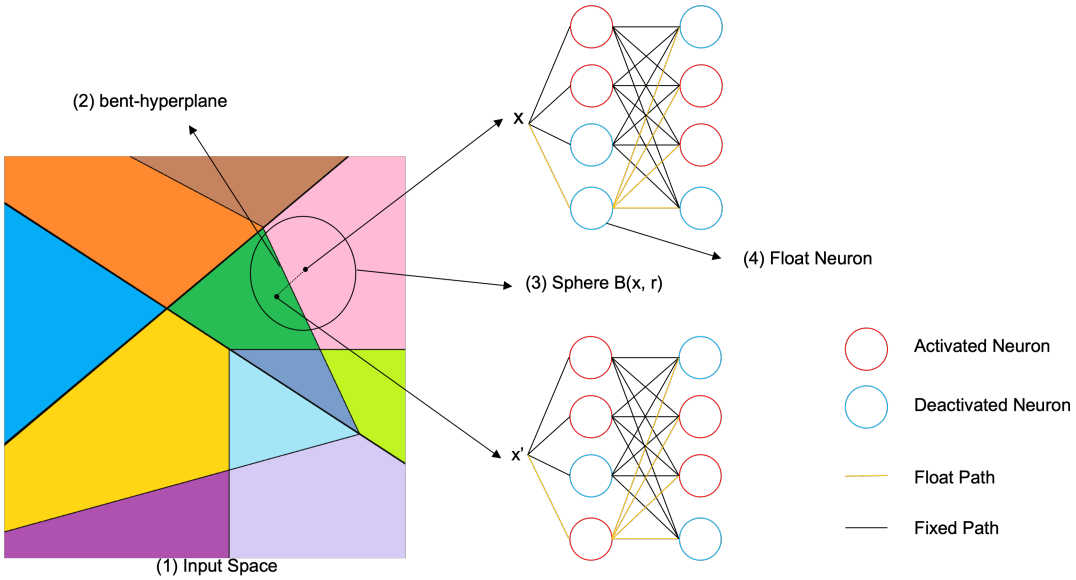


Figure 5.1: To illustrate the concept of pattern similarity and transitino density, this figure presents the input space splitted into several activation regions of by a 2 layers neural network, each layer with 4 neurons.

by the figure.

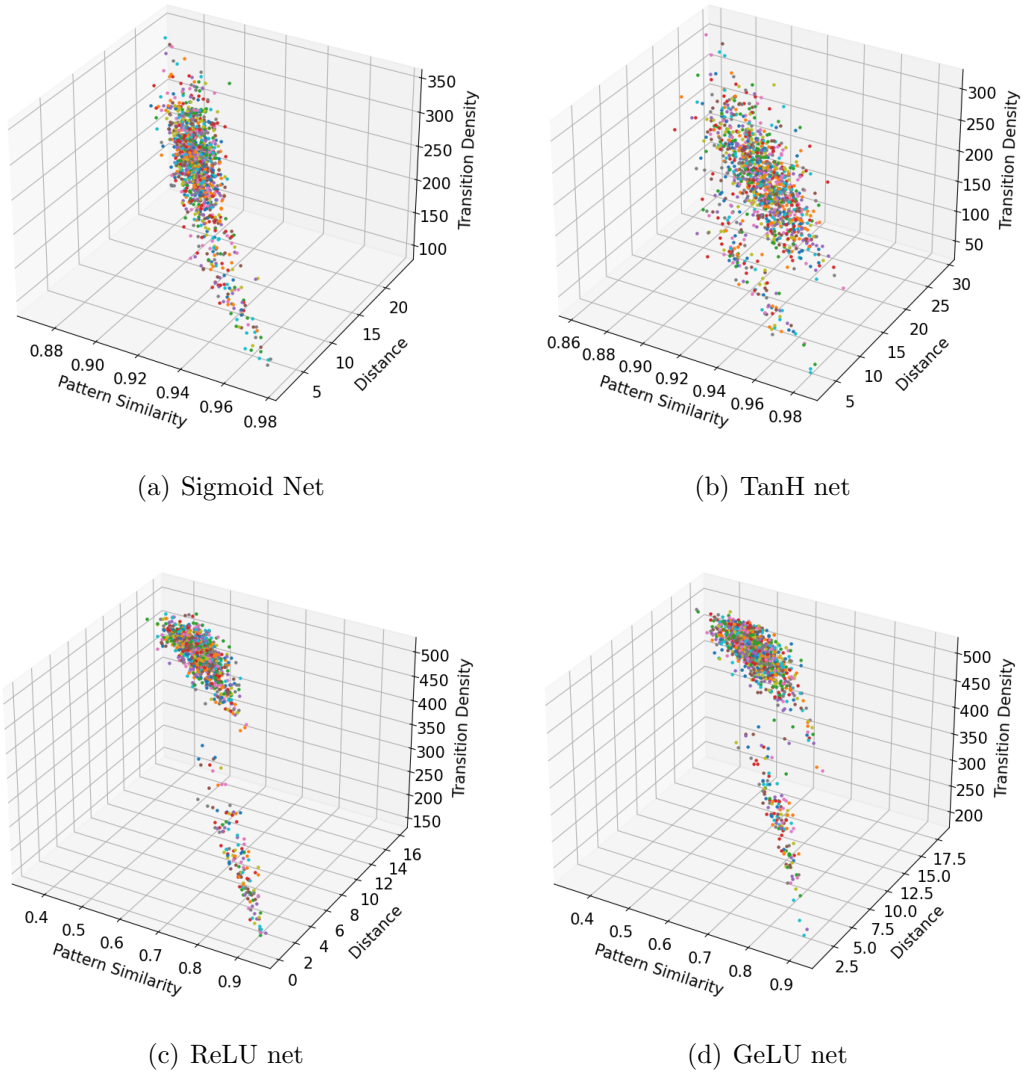
Both of those two metrics are proposed to evaluate the distance of mapping functions for x and x' given the same neural network to simplify the analysis of deep neural networks. In the next section, we use experiments to illustrate the usage of our metrics.

5.2 Experiments and Discussion

This section presents experiments regarding the evaluation of model expressive ability using pattern similarity metrics. The first set of experiments in Section 5.2.1 illustrates properties of the pattern similarity metric concerning Theorem 4. The second part of the experiment aims to answer the introductory question by evaluating models with different activations using our metric. It is found that the *dying neuron* issue widely exists, which is why networks have practically less expressive ability than theoretically expected. Moreover, the severity of the *dying*

neuron issue can explain the differences in model performance.

5.2.1 Evaluating Pattern Similarity



(a) Sigmoid Net

(b) TanH net

(c) ReLU net

(d) GeLU net

Figure 5.2: Distance of predictions, transition density and pattern similarity of 1225 pairs of data.

In Chapter 5.1, a purely theoretical analysis is presented to discuss the connection between pattern similarity and the prediction difference $\delta(\mathbf{x}, \mathbf{x}')$ between data points. To provide empirical support for the result, this section illustrates the metrics and

$\delta(\mathbf{x}, \mathbf{x}')$ in a 3-D plot. Figure 5.2 shows the relationships between pattern similarity, transition density, and prediction distance for models with Sigmoid, Tanh, ReLU, and GeLU activations.

5.2.1.1 Experiment Settings

This set of experiments applies a similar training setup as in Chapter **??**. The neural network used in this section is a stacked fully connected neural network with 784-dimensional inputs, 9 hidden layers with 256 neurons in each layer, and 10-dimensional outputs. The network is trained on the MNIST dataset for 24,000 iterations with a batch size of 128 using the SGD optimizer and a linear rate scheduler decaying from 0.1 to 0.0001. The cost function used in this section is the cross-entropy loss, which is widely adopted in classification tasks.

To analyze the relationship between activation pattern, prediction distance, and transition density, the first 50 data points of the test dataset are selected and combined pairwise, yielding 1225 pairwise data combinations.

On the z-axis, the transition density between two data points is presented. Given $\mathbf{x}, \mathbf{x}'$ from the test dataset, 512 points are sampled from the segment $\overline{\mathbf{x}\mathbf{x}'}$:

$$\mathbf{x}_i = \frac{i}{513}(\mathbf{x}' - \mathbf{x}) + \mathbf{x}, i = 1, 2, \ldots, 512. \tag{5.20}$$

With $\mathbf{x}$ and $\mathbf{x}'$, the transition is calculated sequentially for the 514 data points as defined by Equation 5.2. The transition density is the sum of all the transitions.

The prediction distance and pattern similarity of the paired data are presented on the x-axis and y-axis, respectively. The Euclidean distance is applied to evaluate the prediction difference. Note that during the discussion in Chapter 5.1, the distance metric was not specified, therefore it does not affect the results. The computation of pattern similarity follows the definition proposed in Chapter 4.2. Given a set of breakpoints, each neuron is assigned a sign according to the pre-activation value as described by Equation 4.8 for the data $\mathbf{x}$ and $\mathbf{x}'$. The similarity $\mathbf{PS}(x_1, x_2; \pi, \theta, \Gamma)$ of the patterns $\hat{\mathcal{A}}(\mathbf{x})$ and $\hat{\mathcal{A}}(\mathbf{x}')$ is then defined as the proportion of neurons that have the same sign for both $\mathbf{x}$ and $\mathbf{x}'$ as per Equation 5.3.

The 3D plot illustrates the relationship between pattern similarity, transition density, and prediction difference. In particular, since there are 10 classes in the MNIST dataset, data pairs with the same and different labels are contained in the 1,225 pairs. This also provides insights into how networks with different activation functions separate data.

5.2.1.2 Choice of Breakpoints

The activation pattern is proposed to describe the regional mapping relationship of a network. For continuous activation functions, a good separation Γ should split the domain into regions such that the properties of the activation vary within that region.

For the Sigmoid-net and Tanh-net, the breakpoints are $\Gamma_1 = -0.5, 0, 0.5$, which partition the activation functions into two deactivated regions and one semi-linear region. For the ReLU-net and GeLU-net, the separation is set to be $\Gamma_2 = 0$, which partitions the activation function into one deactivated region and one activated region.

5.2.1.3 Pairwise Pattern Similarity, Transition Density, and Prediction Difference

Generally, the samples are clustered into two strips facing the upper left. The points in the upper space mostly consist of paired data with different labels, which have higher transition density, higher prediction distance, and lower pattern similarity, while the points in the lower space show the opposite. According to the axis tickers, it can be concluded that:

- The transition density is positively related to the prediction difference between data.
- The pattern similarity is negatively related to the transition density.
- The pattern similarity is negatively related to the prediction difference.

The results are in accordance with the theoretical analysis of Theorem 3 and Theorem 4, which can be viewed as empirical support for the results.

The next insight provided by the plot is how Sigmoid-net and Tanh-net differ from ReLU-net and GeLU-net. The two clustered strips are clearly separated for ReLU-net and GeLU-net, suggesting that data with different labels and data with the same labels have distinguishably different metric features. However, for the Sigmoid-net and Tanh-net, the data points are interspersed, meaning that those networks fail to provide a clear separation for different data.

Additionally, it is found that the pattern similarity of Sigmoid-net and Tanh-net has a limited range and is relatively higher, suggesting there are fewer activation regions for those networks. As the number of activation regions relates to the expressive power of the network, Sigmoid-net and Tanh-net have a worse ability to approximate complex functions.

5.2.2 Pattern Similarity on Sub-datasets

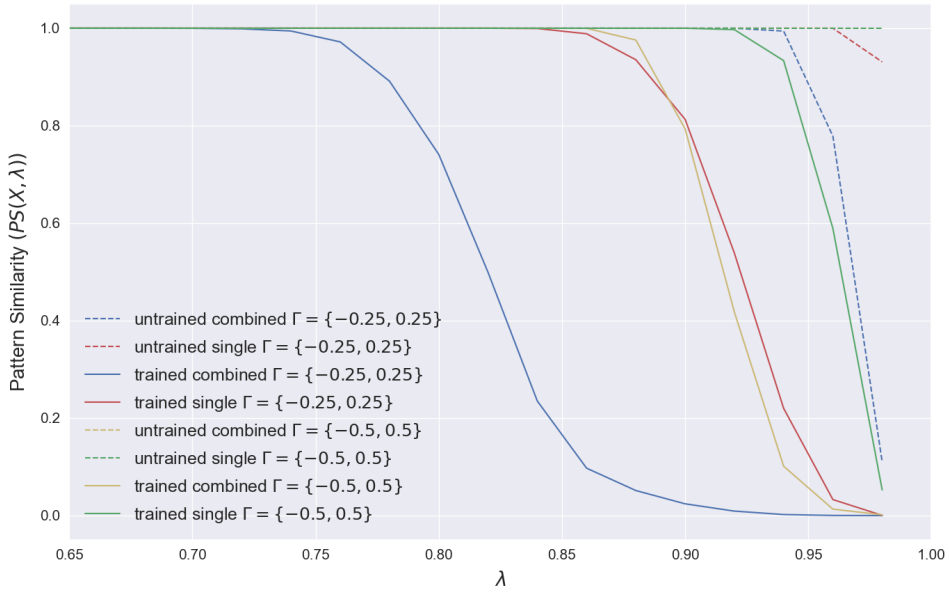
The next step is to understand how the pattern similarity of the model varies across different data distributions. Specifically, this section examines how pattern similarity changes after training.

5.2.2.1 Experiment Settings

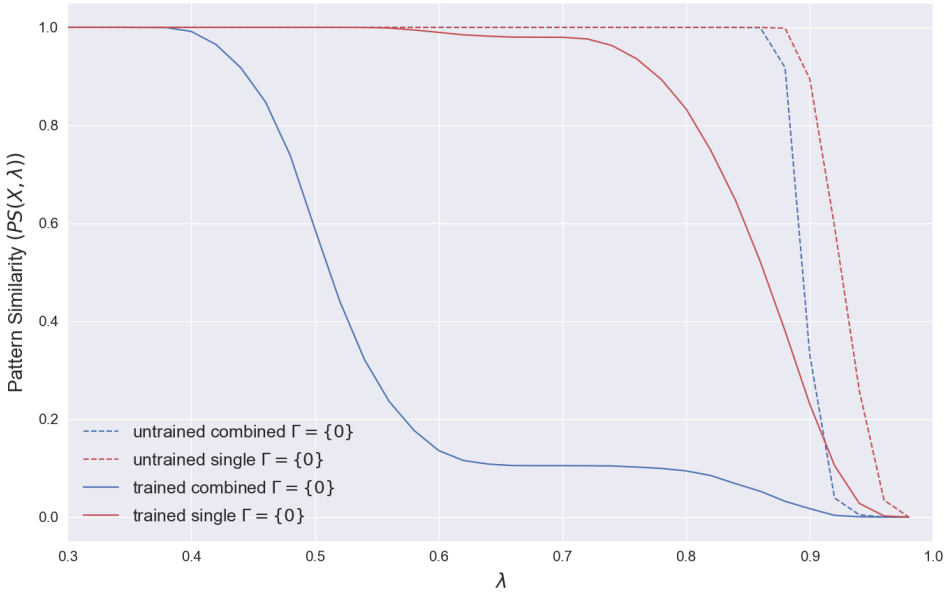
As discussed above, given a well-performing model $\mathcal{N}$ and two data points $\mathbf{x}$ and $\mathbf{x}'$, the pattern similarity should be lower if $\mathbf{x}$ and $\mathbf{x}'$ have the same label, ensuring a tighter bound on $\delta(\mathbf{x}, \mathbf{x}')$, and conversely higher for $\mathbf{x}$ and $\mathbf{x}'$ with different labels.

To evaluate the model's performance, two datasets are constructed from the test data of the MNIST dataset. The *single* dataset is constructed by selecting 1,000 samples with the same label, while the *combined* dataset consists of 1,000 samples randomly selected from the test dataset. We compare the pattern similarity of the two datasets on trained and untrained models.

The model structure and training schedule are the same as those in Section



(a) Pattern Similarity of Sigmoid net



(b) Pattern Similarity of ReLU net

Figure 5.3: Pattern Similarity of ReLU Net and Sigmoid Net on Different Datasets.

5.2.1. The separation of activation functions is set to be $\Gamma_{s_1} = -0.25, 0.25$ and $\Gamma_{s_2} = -0.5, 0.5$ for the Sigmoid-net, and $\Gamma_r = 0$ for the ReLU-net.

5.2.2.2 Indicators of Expressive Ability.

Figure 5.3 presents the experiment results for the Sigmoid-net and ReLU-net. The gap between the pattern similarity of the *single* dataset and the *combined* dataset can be viewed as an indicator of the practical model's expressive ability.

As discussed in Section 5.2.1, lower pattern similarity between data implies higher prediction differences. By generalizing this observation to a dataset, it can be inferred that the pattern similarity within a dataset should be low if the majority of the data have different labels. In other words, since pattern similarity measures the neuron-level response towards a dataset, a model with high expressive ability should be able to distinguish the differences between data with different labels, thereby showing low pattern similarity for the *combined* dataset and the opposite for the *single* dataset.

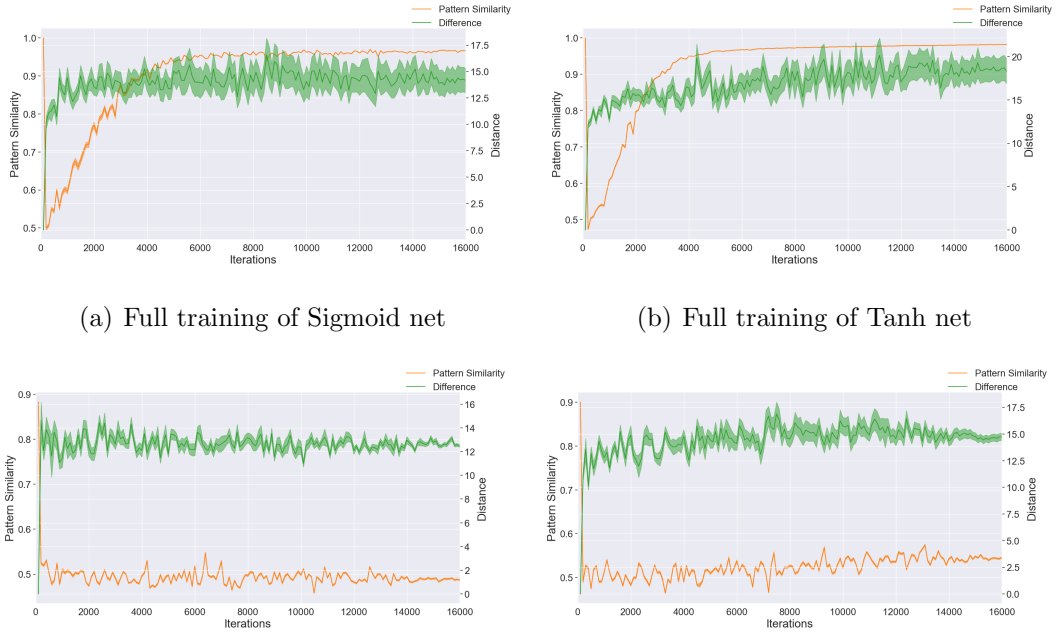
The dashed lines show the pattern similarity of the untrained models on different datasets. For both models, the dashed lines start to decline at around $\lambda = 0.90$, suggesting that the activation patterns of the dataset are around 90

The blue solid line in Figure 5.3(a) shows that after the model is trained, the pattern similarity for the *combined* dataset decreases dramatically compared to the blue dashed line. This means that the model is able to capture the essential features of data with different labels. Meanwhile, for the red solid line, the pattern similarity remains at a high level until λ reaches 0.7, indicating that for data with the same label, around 70

In contrast, Figure 5.3(a) shows that the expressive ability of the Sigmoid-net is relatively worse. For both separations, the gap between the *combined* and *single* datasets is less than 10%. Specifically, the solid lines remain at 1 until λ reaches around 0.8, which means that for any input data, 80% of neurons yield similar post-activation values. In other words, neurons fail to learn the distinctive features of data with different labels. This results in a novel *dying neuron* issue where, even though most neurons are activated, their post-activation values remain in the same region for data with different labels, thereby failing to provide useful information.

5.2.3 Expressive Ability during Training

This section investigates the *dying neuron* issue during the training of neuron networks to compare the expressive ability of network with different activations.



(a) Full training of Sigmoid net

(b) Full training of Tanh net

(c) Full training of ReLU net

(d) Full training of GeLU net

Figure 5.4: Pattern similarity, prediction distance of fully connected network with different activations during the training. The network consists of 9 layers fully connected network with 256 neurons within each layer.

5.2.3.1 Experiment Settings.

This section considers both stacked fully connected neural networks (FCNN) and convolutional neural networks (CNN). The structure and training schedule for the FCNN is the same as introduced in Chapter **??**. The CNN used in this work is the VGG16 network [91], trained on the CIFAR10 dataset for 120 epochs with a batch size of 128 and an SGD optimizer with a linear decay learning rate from 0.1 to 0.0001. The pattern similarity and prediction distance are computed using a fixed set of 1,000 pairs of test data with different labels.

General Performance. We first illustrate the general performance of the models. The metrics are recorded every 100 iterations for the FCNN and every epoch for the VGG16 network.

Figure 5.4 shows the change in pattern similarity and prediction distance for the FCNN. For the Sigmoid-net and Tanh-net, the pattern similarity first decreases and then gradually increases to around 0.95. This indicates that during training, most of the neurons in neural networks with saturated activations are gradually pushed to a region where they exhibit the same response regardless of the input. Moreover, the neurons are unable to escape from such a region. This result aligns with previous discussions [17].

In contrast, for networks with ReLU or similar activations, the pattern similarity reduces to around 0.5 at the very beginning of training and remains unchanged thereafter. This suggests that for data with different labels, around 50% of the neurons react differently to the input. In other words, these neurons are able to distinguish the differences between data from different classes.

Apart from that, the variance of the prediction difference for neural networks with different activations is also a good indicator of model stability. In fact, it can be observed that the variance of $\delta(\mathbf{x}, \mathbf{x}')$ exhibits similar behavior to the gradients of models during training, as presented in Figures **??**, **??**, and **??**. ReLU has the most stable gradients during training as well as the lowest variance of $\delta(\mathbf{x}, \mathbf{x}')$, followed by GeLU, Tanh, and Sigmoid. This explains why the model performance of the Sigmoid-net is the least stable.

5.2.3.2 VGG16

Figure 5.5 shows the change in pattern similarity of the VGG16 network with different activations. During the training, the pattern similarity of the GeLU and ReLU networks is relatively more stable. By the end of the training, the GeLU net has the lowest pattern similarity for data with different labels, implying that the GeLU net has better expressive ability.

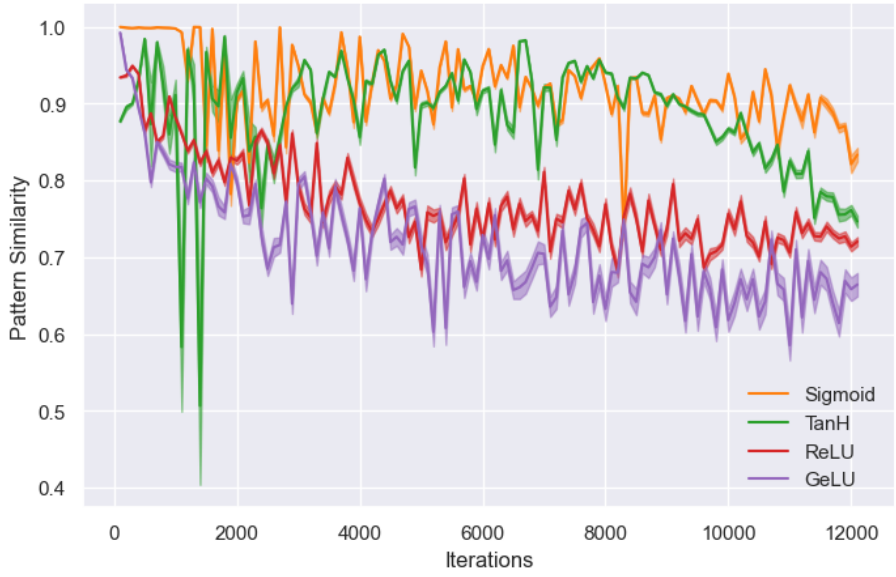


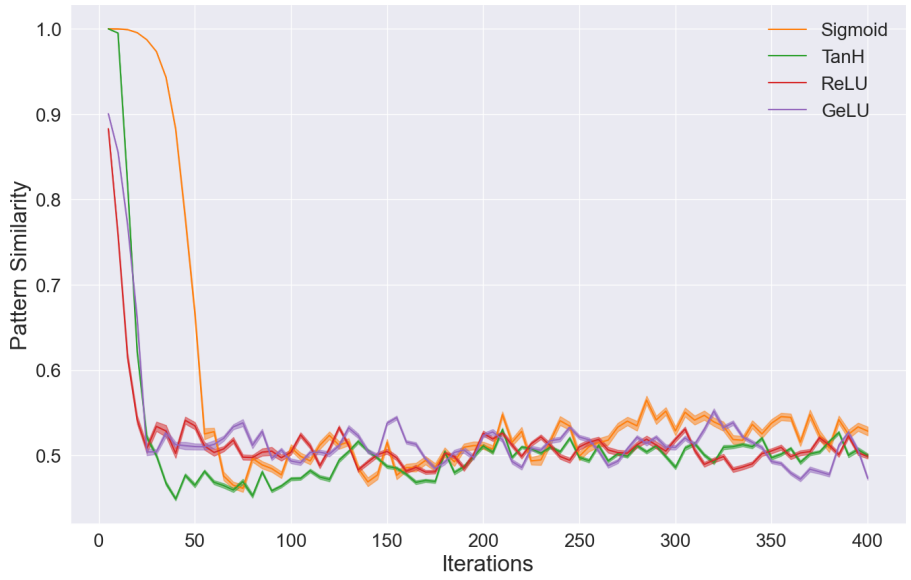
Figure 5.5: Pattern Similarity of VGG16 network with different activations.

5.2.3.3 Early Stage of Training.

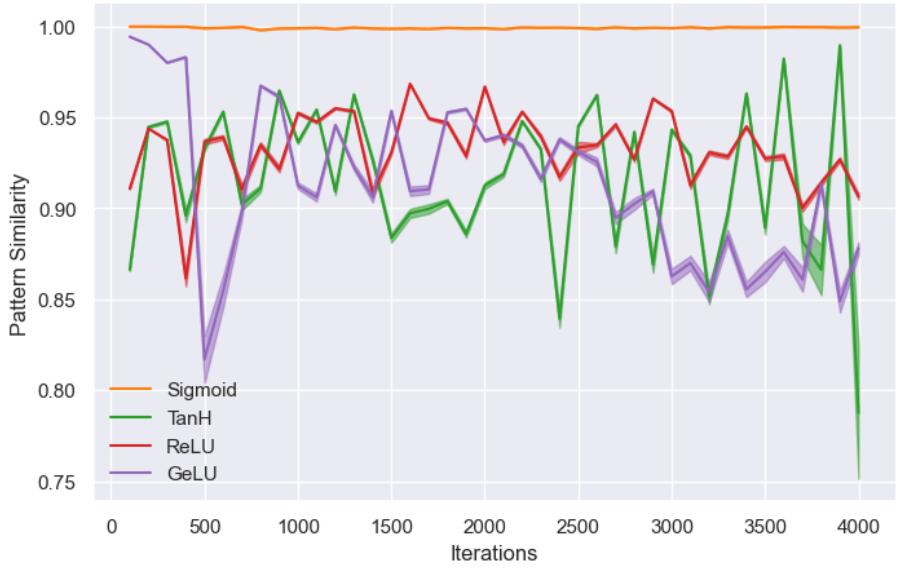
Another helpful measurement is the convergence speed of networks with different activation functions. Figure 5.6 presents the behavior of pattern similarity during the early stage of training. For both the FCNN and VGG16 networks, the pattern similarity is recorded every 5 iterations.

Figure 5.6(a) shows the results for the FCNN network. For all the models, the pattern similarity gradually decreases to around 0.5. The ReLU net has the fastest convergence speed, followed by the GeLU net and Tanh net. This result aligns with the previous discussion about the training dynamics of models in Chapter 3. However, the Sigmoid net is shown to converge much slower.

This phenomenon is even more pronounced in a larger network. Figure 5.6(b) presents the change in pattern similarity during the early stage of training for the VGG network. The pattern similarity of the Sigmoid net remains at 100% at the very beginning of training, indicating that the network fails to learn valid information at the start of the training process.



(a) FCNN



(b) VGG16

Figure 5.6: Pattern Similiarity of fully connected network with different activations at early stage of training. The network consists of 9 layers fully connected network with 256 neurons within each layer.

	MNIST	**CIFAR10**
Model	FCNN	VGG16
Sigmoid	98.28	82.45
Tanh	98.25	84.30
ReLU	98.45	92.27
GeLU	*99.17*	*92.73*

Table 5.1: Test accuracy of network different activations.

5.2.3.4 Performance

The expressive ability provided by our metrics is directly related to model performance. Table 5.1 shows the test accuracy of different networks. The GeLU activation has the highest test accuracy for both networks, with 99.17% on MNIST and 92.73% on CIFAR10, followed closely by the ReLU net with slightly lower accuracy. On the other hand, the Sigmoid-net and Tanh-net have lower test accuracy. Notably, when combined with the results from Figure 5.5, it can be observed that the accuracy ranking mirrors the pattern similarity ranking. Since pattern similarity is an indicator of the severity of the *dying neuron* issue, the performance gap between models can be explained by this issue.

5.3 Expressive Ability and Neuron Entropy

The discussion in the previous section addresses the introductory question of why the Sigmoid net has lower generalization ability and accuracy. This section, rather than analyzing performance across different models, aims to bring insights into the activation pattern within a single model.

Given a data distribution $\mathcal{D}$ with c classes, this section investigates the relationship between model performance and neuron behavior on the support of the distribution $\mathcal{D}$. It introduces a *neuron entropy* metric and describes its connection

with model performance. The proposed metric measures the instability of a neural network on a dataset in a manner similar to that of *pattern similarity*, but it provides neuron-level information about the network. This leads to an in-depth discussion of how expressive ability is affected by the neural network, as well as a neuron entropy pruning (NEP) method that removes insensitive neurons to reduce the scale of the network.

5.3.1 Neuron Entropy

The discussion in the previous section shows that, given a neural network $\mathcal{N}$ and a data distribution $\mathcal{D}_x$ with different labels, the expressive ability of $\mathcal{N}$ is negatively related to its pattern similarity on the dataset. Specifically, low pattern similarity means that neurons are proactively responding to data with different activation statuses, and therefore have more potential in approximating complex objective functions. This suggests that neurons should have more diverse activation patterns on a dataset to represent complex functions.

To better understand this, consider the following extreme case. Assume $\mathcal{N}$ is a neural network with piecewise linear activation, and all the neurons of the neural network $\mathcal{N}$ on the support of distribution $\mathcal{D}$ are fixed: $\mathcal{I}^T(\text{supp}(\mathcal{D})) = \emptyset$. In this case, the function of $\mathcal{N}$ is a linear function. This suggests that a network with an insufficient number of float neurons has a diminished ability to approximate functions with high complexity. As the number of float neurons increases, the expressive ability of the network $\mathcal{N}$ also increases.

Motivated by this intuition, we investigate the float neurons of the neural network defined on the input space. The input space is the union of numerous activation regions, where the activation patterns in each region differ. Specifically, Section 5.3.3.2 shows that for most neurons in $\mathcal{N}$, there exist points $\mathbf{x}$ and $\mathbf{x}'$ such that the activation patterns on $\mathbf{x}$ and $\mathbf{x}'$ are different. This implies the difficulty of connecting model expressive ability with neuron stability. Therefore, an explicit measure of neuron stability under a global scope is necessary to continue our study.

We introduce neuron entropy as follows.

5.3.1.1 Neuron Entropy

Definition 6 (Neuron Entropy)**.** *Let $\mathcal{N}$ be a network as defined in Chapter 4.1. The entropy of a neuron (i,j) in $\mathcal{N}$ for a data distribution $\mathcal{D}$ is defined as the entropy of the distribution of the activation status for (i,j) given input $(\mathbf{x},y) \in \mathcal{D}$.*

$$\mathcal{E}_j^{(i)}(\mathcal{N},\mathcal{D}) = -\sum_{k=0}^{q} p(\hat{a}_j^{(i)}(\mathbf{x}) = k)\log(p(\hat{a}_j^{(i)}(\mathbf{x}))). \tag{5.21}$$

The neuron entropy describes the uncertainty of the neuron activation pattern. For instance, let $\mathcal{N}$ be a neural network with ReLU activation. If $\mathcal{E}_j^{(i)}(\mathcal{N},\mathcal{D}) = 0$, this suggests that the activation pattern of neuron (i,j) is identical for any data $(\mathbf{x},y) \sim \mathcal{D}$, implying that neuron (i,j) fails to provide non-linearity. On the other hand, if $\mathcal{E}_j^{(i)}(\mathcal{N},\mathcal{D})$ is close to 0.69, it indicates that the probability of neuron (i,j) being either activated or deactivated is around 50% for data distribution $\mathcal{D}$.

Figure 5.7 shows the neuron entropy of neuron $\hat{a}_j^{(i)}$, where the x-axis represents the probability that the pattern of $\hat{a}_j^{(i)}$ is 0: $P(\hat{a}_j^{(i)} = 0)$. For activations with two patterns (such as ReLU, PReLU), neuron entropy reaches its maximum of around 0.69 when $P(\hat{a}_j^{(i)}(\mathbf{x}) = 0) = 0.5$. For activations with three patterns (such as Sigmoid, ReLU-6), the maximum entropy is 1.09 when $P(\hat{a}_j^{(i)}(\mathbf{x}) = 0) = 1/3$.

Now we build the connection between neuron entropy and model expressive ability. Let $(\mathbf{x},y),(\mathbf{x}',y') \sim \mathcal{D}$ with $y \neq y'$. To perform correct classification, network $\mathcal{N}$ should be able to describe the difference between $\mathbf{x}$ and $\mathbf{x}'$. Theorem 8 decomposes $f(\mathbf{x}) - f(\mathbf{x}')$ into a linear part derived from the Jacobian matrix and a non-linear part. The linear part is determined by the local Jacobian matrix at $\mathbf{x}$. The non-linear part, on the other hand, is determined by the number of float neurons in the set $\mathbf{x},\mathbf{x}'$. As the number of float neurons increases, the number of float paths in $\mathbf{x},\mathbf{x}'$ and the complexity of $f(\mathbf{x}) - f(\mathbf{x}')$ also increase. This implies that the network is able to provide better non-linearity and has greater potential in approximating the difference between $\mathbf{x}$ and $\mathbf{x}'$.

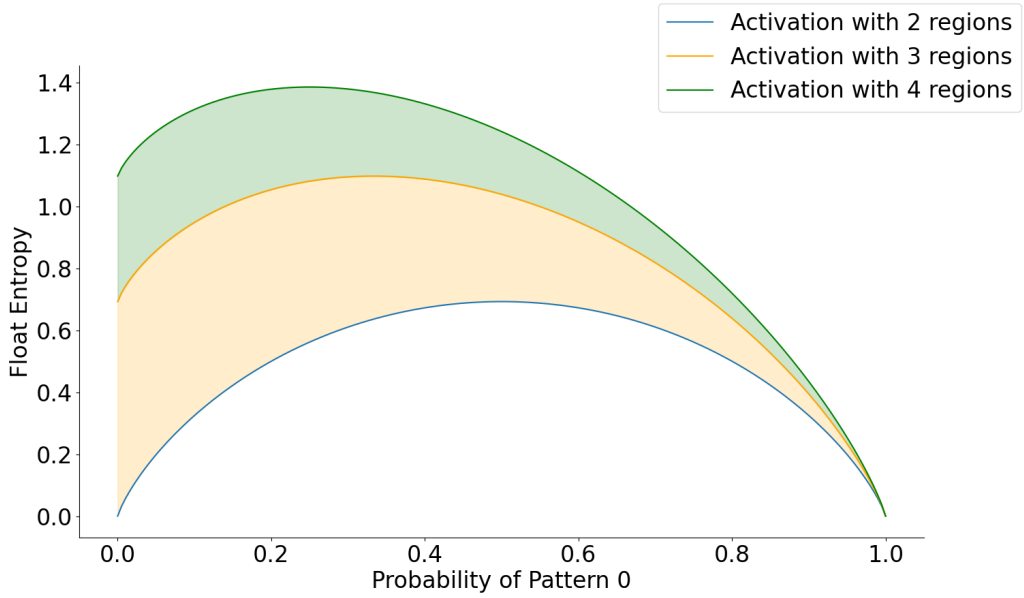


Figure 5.7: The float entropy of activation functions with k different patterns given the probability of pattern is 0 (x-axis). Each line shows the maximum value of entropy, while the shadowed region is the range of entropy from $k = n$ to $k = n - 1$.

To generalize the discussion from data points to data distributions, we introduce neuron entropy, which is built on the distribution of activation patterns and describes how well neurons can provide non-linearity given a data distribution $\mathcal{D}$. In other words, it can be viewed as an indicator representing the expected complexity of the estimation of $f(\mathbf{x}) - f(\mathbf{x}')$. Formally, we use the following theorem to connect model expressive ability and neuron entropy.

Theorem 5. *Consider two neural networks $\mathcal{N}$ and $\mathcal{N}'$ with monotonous activation functions, under the same architecture, and trained on the same data distribution $\mathcal{D}$. Denote the measure zero parameter set with respect to the Lebesgue measure for $\mathcal{N}$ and $\mathcal{N}'$ as θ and θ'. Let $(\mathbf{x}, y), (\mathbf{x}', y') \sim \mathcal{D}$. If:*

1. $P(\mathbf{x} = \mathbf{a}) = 0$, $a \in R^{n_0}$;

2. $\boldsymbol{\theta}, \boldsymbol{\theta}' \sim \Theta$

3. $\sum_{i,j} \mathcal{E}_j^{(i)}(\boldsymbol{\theta}, \mathcal{D}) < \sum_{i,j} \mathcal{E}_j^{(i)}(\boldsymbol{\theta}', \mathcal{D})$,

then $\mathbb{E}(f(\mathbf{x}; \boldsymbol{\theta}) - f(\mathbf{x}'; \boldsymbol{\theta})) \leq \mathbb{E}(f(\mathbf{x}; \boldsymbol{\theta}') - f(x'; \boldsymbol{\theta}'))$.

Proof. For any $x, x' \in R^n$, consider the distance between $f(\mathbf{x}; \theta) - f(\mathbf{x}'; \theta)$. According to Lemma 8:

$$f(\mathbf{x}) - f(\mathbf{x}') = J(x)(\mathbf{x} - \mathbf{x}') + Z^T(x', \mathcal{A}; \mathcal{R}) - Z^T(\mathbf{x}', \mathcal{A}; \mathcal{R}) \tag{5.22}$$

□

The above theorem compares the expressive abilities of neural networks with the same structure but different parameter sets. Among the assumptions: (1) shows that there is no atom from the data distribution, which is trivial and compliant with any continuous probability function; (2) assumes that the parameter sets of networks $\mathcal{N}$ and $\mathcal{N}'$ are drawn from the same distribution, which is empirically satisfied when networks are trained under the same algorithm; and (3) suggests that network $\mathcal{N}$ has lower average entropy compared to network $\mathcal{N}'$.

The conclusion of Theorem 5 compares the expected ability of models to separate $(\mathbf{x}, y), (\mathbf{x}', y') \sim \mathcal{D}$. For a data distribution with more than two classes, the probability of $\mathbb{P}(y = y') > \mathbb{P}(y \neq y')$ is higher than that of $y \neq y'$. Therefore, better ability in separating $f(\mathbf{x})$ and $f(\mathbf{x}')$ implies a higher expressive ability of the network. In Section 5.3.3, we present empirical results of our metric in analyzing neural network performance.

5.3.2 Model Expressive Ability

This section discusses the link between neuron entropy and metrics proposed in previous works.

The number of linear regions was one of the earliest metrics introduced to quantify model expressive ability. Let $\mathcal{N}$ be a neural network with a piecewise linear activation function. The input space is divided into numerous linear regions within which the mapping is a linear function. Therefore, the number of linear regions of a

network is viewed as a representation of network complexity. The following theorem shows that higher average neuron entropy implies more linear regions on the support of distribution $\mathcal{D}$.

Theorem 6. *Let $\mathcal{N}$ and $\mathcal{N}'$ be neural networks with the same structure as defined in Chapter 4.1. Denote the parameter set of $\mathcal{N}$ and $\mathcal{N}'$ as $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$. If $\sum_{i,j} \mathcal{E}_j^{(i)}(\boldsymbol{\theta}, D) > \sum_{i,j} \mathcal{E}_j^{(i)}(\boldsymbol{\theta}', D)$, then*

$$\#\textit{Linear Regions of } \mathcal{N} > \#\textit{Linear Regions of } \mathcal{N}'.$$

The connection between neuron entropy and the number of activation regions is built upon the probability of two random points $\mathbf{x}, \mathbf{x}' \sim \mathcal{D}$ being within the same region. A higher average neuron entropy of a network implies a lower probability that $\mathbf{x}$ and $\mathbf{x}'$ share the same activation pattern. Conversely, this lower probability suggests that there are more activation regions.

Subsequent work introduces transition density as a metric for evaluating the stability of a neural network. Transition density describes the number of activation regions crossed by a trajectory. A higher transition density implies that the network is less stable along that trajectory.

In evaluating model expressive ability, [2] introduces a pattern similarity metric that measures the ratio of neurons with the same activation pattern given a pair of data from the dataset:

$$\mathbf{PS}(D; \boldsymbol{\theta}) = \mathbb{E}_{x,x' \sim D}[\frac{\mathcal{I}^X(\mathbf{x}, \mathbf{x}')}{\#\text{number of neurons in } N}] \tag{5.23}$$

As is discussed in [2], higher pattern similarity is associated with a lower transition density as well as a loss of model expressive ability. A dying neuron issue is also observed that most of the neurons are firing similar signals regardless of the input. The following theorem describes the connection between our metric and pattern similarity.

Theorem 7. *Let $\mathcal{N}, \mathcal{N}'$ be neural networks with the same structure and piece-wise linear activation function. Denote the parameter set of $\mathcal{N}$ and $\mathcal{N}'$ are $\boldsymbol{\theta}$ and*

$\boldsymbol{\theta}'$. *Given data distribution* $\mathbb{D}$, *if* $\mathbb{E}(\mathcal{E}_j^{(i)}(\boldsymbol{\theta}, \mathbb{D})) > \mathbb{E}(\mathcal{E}_j^{(i)}(\boldsymbol{\theta}', \mathbb{D}))$, *then* $\mathbf{PS}(\mathbb{D}; \boldsymbol{\theta}) > \mathbf{PS}(\mathbb{D}; \boldsymbol{\theta}')$.

Theorem 6 and Theorem 7 connect the metrics proposed in previous works with neuron entropy, demonstrating the consistency of the generic results of our metric. On the other hand, neuron entropy offers significant advantages in evaluating the performance of a neural network from a unit-wise perspective. For instance, comparing neuron entropy across different layers can shed light on the role of each layer, while monitoring changes in unit-wise neuron entropy allows us to identify *dead neurons*.

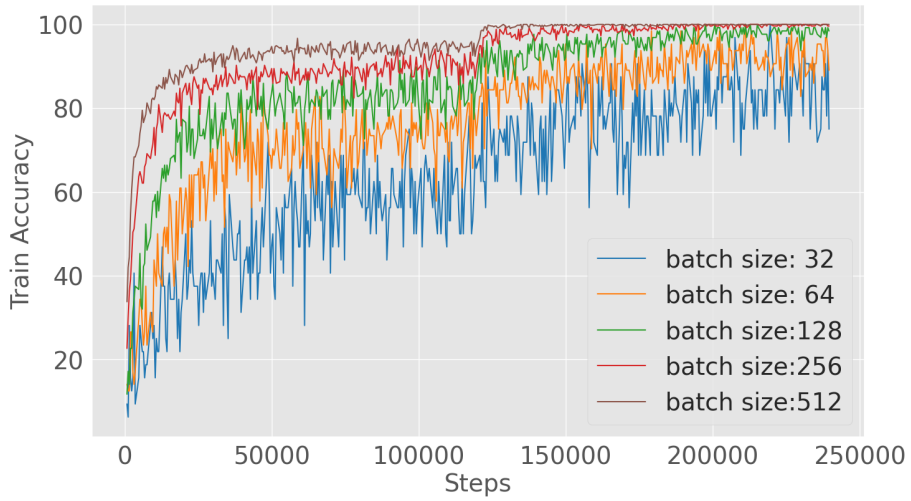
5.3.3 On Explaining Neural Networks

5.3.3.1 Model Expressive Ability

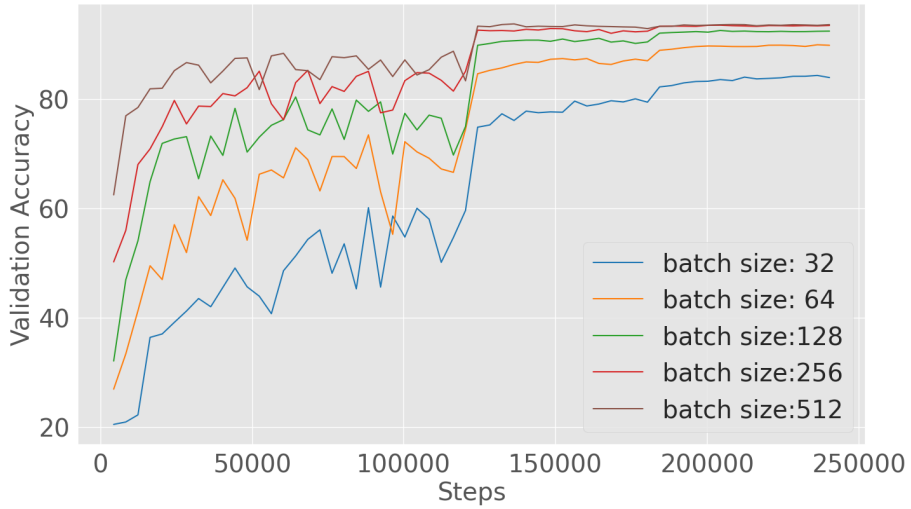
Figure 5.8 compares the neuron entropy of VGG16 networks trained on the CIFAR10 dataset with different batch sizes. Each model is trained for 24,000 steps, and the neuron entropy is evaluated every 400 steps. The activation function for all networks is ReLU, with the separation 0. Therefore, each neuron in the network is either activated or deactivated.

Figures 5.8(a) and 5.8(b) compare the training and validation accuracy of different models. The results show that models with larger batch sizes tend to have higher accuracy and less instability in both training and testing datasets. To explain the performance gap from a neuron stability perspective, Figures 5.9(a) to 5.9(c) compare the average neuron entropy at different layers, while Figures 5.9(d) to 5.9(f) compare the variance of neuron entropy at different layers.

At the entry layer (layer 1, Figure 5.9(a)), the neuron entropy of different models is close to 0.62, indicating that most of the neurons have around a 50% probability of being activated for a given dataset. In the shallow layers (layer 6, Figure 5.9(b)), the neuron entropy first increases to around 0.6 and then gradually drops. However, the neurons in the best-performing model (with batch size 512, brown line) are more



(a) Training Accuracy

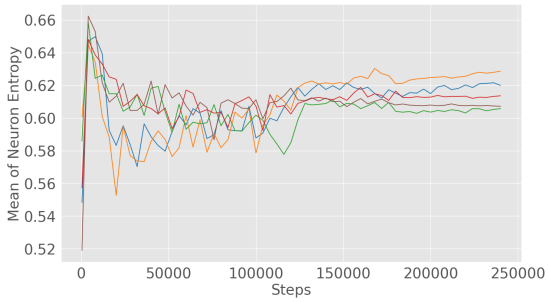


(b) Test Accuracy

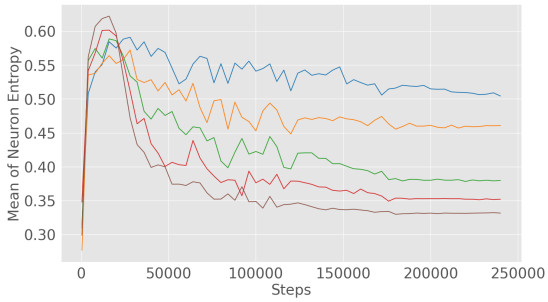
Figure 5.8: The mean and variance neuron entropy at different layers for a VGG16 network trained on CIFAR10 dataset.

stable in the shallow layers.

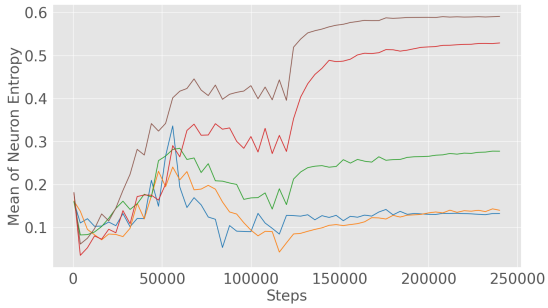
The most distinguishable differences between models can be observed in the deep layers (layer 11, Figure 5.9(c)). It shows that the model with higher neuron entropy is able to provide better accuracy on both training and testing datasets. In particular, the neuron entropy of the worst-performing models is around 0.1, which means that most of the neurons in the deep layers have the same activation status



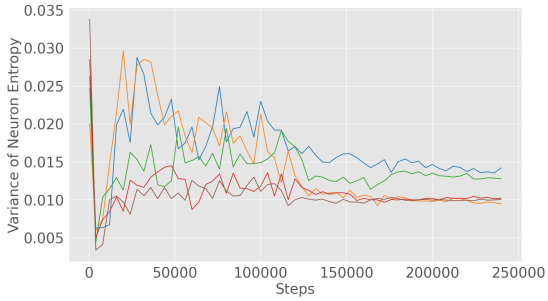
(a) Mean of Layer 1



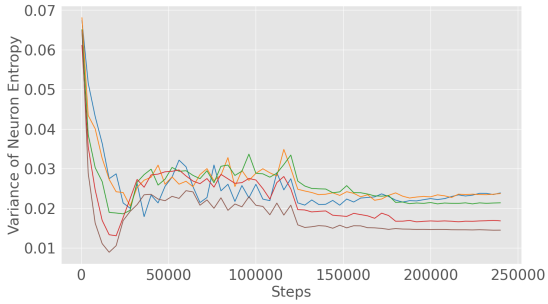
(b) Mean of Layer 6



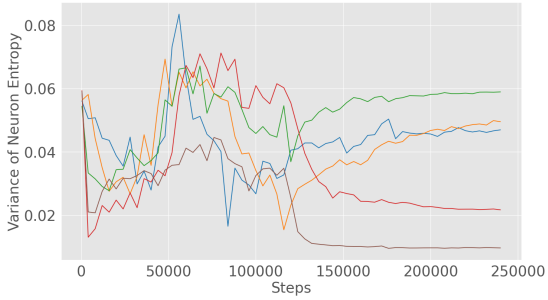
(c) Mean of Layer 11



(d) Variance of Layer 1



(e) Variance of Layer 6



(f) Variance of Layer 11

regardless of the input. This is known as the *dying neuron* issue, which causes a loss of expressive ability [2].

On the other hand, by comparing the variance of neuron entropy, we find that at different layers, the best-performing model has the lowest entropy variance. This means that all the neurons have similar uncertainty at each layer.

The comparison of neuron entropy at different layers helps explain neural network performance from different aspects. First, it shows that neuron entropy for a well-performing model should be higher in the deep layers while lower in the shallow

layers. Second, it reveals the role of each layer. At the entry layer, the neurons are highly unstable due to the diversity of input images. The shallow layers, as suggested by previous works, are responsible for generating features from the input, and therefore tend to have stable neurons. Prior to classification, the deep layers then extract essential information for classification. The high neuron entropy in the deep layers indicates a better ability to encode features for classification.

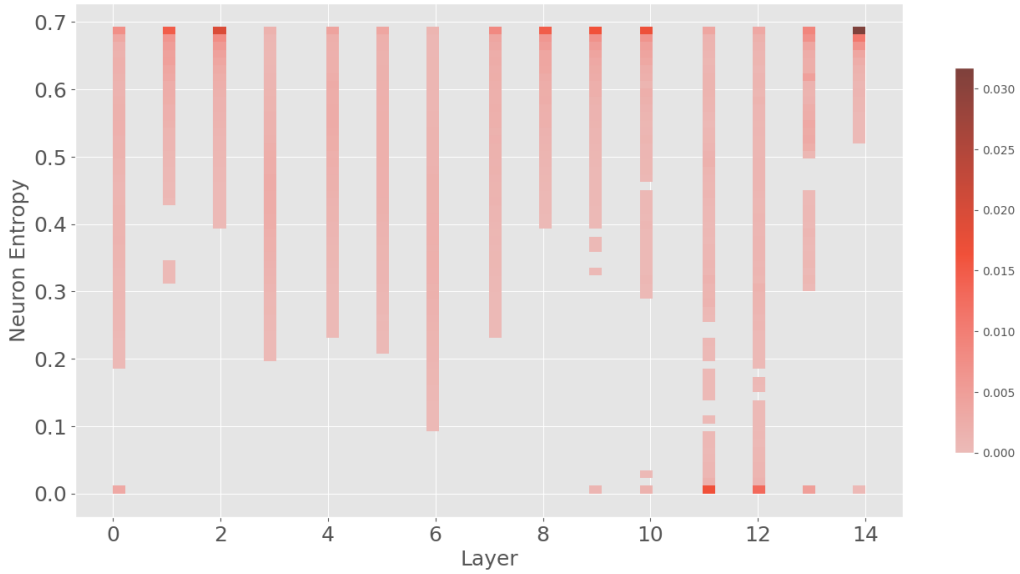
5.3.3.2 Overfitting and Dying Neuron

This section discusses the generalization of neural networks. We train VGG16 networks on the CIFAR10 dataset for 120 epochs with a batch size of 256 using the SGD optimizer. A milestone learning rate scheduler is adopted, where the learning rate starts at 0.1 and is multiplied by 0.1 after 60 and 90 epochs. The training and validation accuracy are 99.7

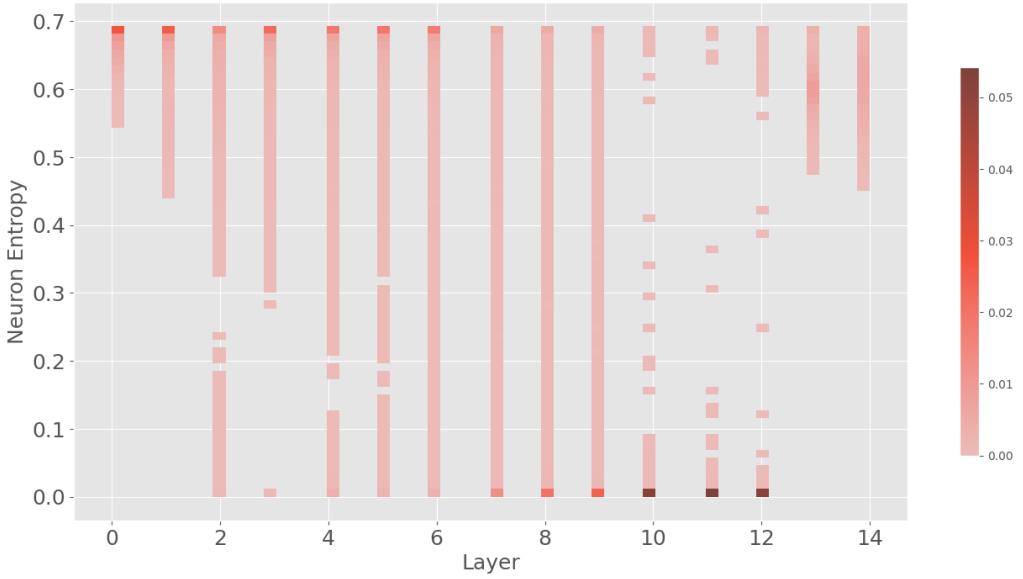
Figure 5.9 illustrates the distribution of neuron entropy on the training and validation datasets at different layers. On the training set, the neuron entropy values of the network are evenly distributed in the entry and shallow layers. In the deep layers, we observe an increase in the proportion of neurons with entropy close to 0. This distribution aligns with the results from Figure 5.8.

On the validation set, the distribution of neuron entropy in the first few layers is relatively more diverse than on the training set. However, starting from layer 7, more neurons become globally stable on the dataset. Notably, from layer 10 to layer 12, a significant proportion of neurons have neuron entropy of 0, meaning these neurons have the same activation status regardless of the input. In other words, they fail to provide any non-linearity for any $\mathbf{x} \sim \mathcal{D}$, resulting in a loss of generalization.

The phenomenon where the activation pattern of neurons remains identical for any input is referred to as the *dying neuron* issue. Our experiments in this section suggest that the generalization error of a neural network can also be attributed to the *dying neuron* issue. Furthermore, by comparing the distribution of neuron entropy on the training and validation datasets, we identify that this issue mainly



(g) Training set



(h) Validation set

Figure 5.9: The average proportion of float neurons for VGG16. The network is trained on the CIFAR10 dataset for 120 epochs. We test the first 1000 data from the validation dataset. For each of the test datasets, we generate a noised dataset with the size of 1000 samples and perturbation size of 4/255 under ℓ^2 bound.

occurs in the deep layers of the neural network, which is also a primary cause of generalization error.

5.3.3.3 Neuron Entropy Dynamics

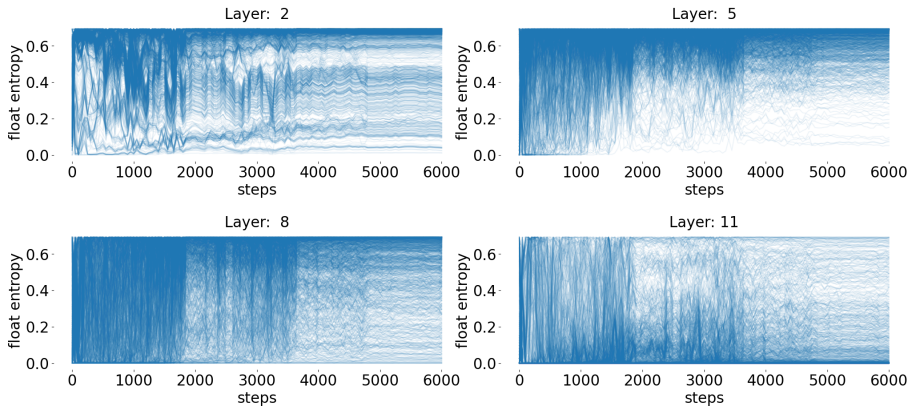


Figure 5.10: Dynamic of neuron entropy at different layers during training. Each of the line records the entropy of a neuron during training every 100 steps.

Figure 5.10 illustrates the dynamics of neuron entropy during training across different layers. The model used is the same as that in Section 5.3.3.2. For each layer, we present the entropy of the first 4096 neurons throughout the training process.

In the entry layers (layer 2), most neurons have an entropy around 0.7, indicating that they are volatile on the dataset and can identify differences across various data points. In the first few layers (up to layer 5), the entropy of neurons increases rapidly during training and stabilizes around 0.7.

However, as the model progresses to deeper layers, more neurons begin to lose their volatility, as shown in layers 8 and 11. For example, by the end of the training, the number of neurons with entropy between 0.2 and 0.4 in layer 8 is higher than that in layer 5. Additionally, in layer 11, most neurons end up with a neuron entropy of 0.

By analyzing the dynamics of neuron entropy, we gain insights into neuron-level

behavior on the dataset. This allows us to identify specific *dead neurons* in different layers that fail to contribute to model performance.

5.4 Pruning of the Dead Neurons

One of the objectives of theoretical analysis in research is to derive insights from observations and introduce innovations to existing models. This section introduces a downstream application inspired by the study of model expressive ability.

We begin by describing the motivation and the connection with the theoretical foundation of the proposed algorithm in Section 5.4.1. This is followed by a discussion of how the proposed method relates to mainstream neural network pruning methods. Finally, we present experimental results demonstrating that the proposed method can outperform benchmark models under the same sparsity conditions.

5.4.1 Motivation

The investigation in the previous chapter focuses on explaining the expressive ability of neural networks and highlights that most networks suffer from the *dying neuron* issue. It reveals that a large proportion of neurons have negligible entropy in a well-trained model, particularly in deeper layers. This analysis aligns with previous discussions on model expressive ability [2], [19]. It shows that despite the potential to represent functions with high complexity, the expressive ability of neural networks is limited due to inactive neurons, referred to as *dead neurons*.

These dead neurons produce similar outputs regardless of the input, contributing little to the prediction process of the neural network. However, during inference, these neurons still consume computational resources. An intuitive approach to improving the performance of large-scale models is to remove these dead neurons, thereby reducing the model's size.

Based on this idea, this section introduces a pruning algorithm called Neuron

Entropy Pruning (NEP). NEP aims to identify the parameters in the neural network that are least important to model performance. It combines neuron entropy with other metrics that evaluate the importance of each parameter. These less important parameters are then removed to reduce the model's size while maintaining its expressive ability. In other words, the objective of NEP is to eliminate the *dead neurons* without compromising the expressive ability of the neural network.

5.4.2 Blending Entropy into Weights

The essence of pruning techniques is to remove unnecessary parameters. In previous works, the importance of parameters is often measured by the absolute value of weights, with the assumption that larger values have a higher impact on prediction results. The objective of NEP is to introduce neuron entropy, which measures the generalization ability of neurons in a global context, into the importance score of a parameter, denoted as $\mathbf{IS}(W^i)$ for the parameters W^i in layer i.

In this context, the weights and neuron entropy can be viewed as prior and posterior measures of the importance of parameters. In the following discussion, they are denoted as:

$$\begin{aligned} &\mathbf{IS}_{prior}(W^i) = W^i \\ &\mathbf{IS}_{post}(W^i) = \text{Importance score derived from neuron entropy.} \end{aligned} \tag{5.24}$$

Given input $\mathbf{x}^{(i)}$, outputs $y^{(i)}$, and weights $W^{(i)}$ of layer i, the roles of weights and neuron entropy can be described as follows:

- During forward propagation, A parameter $W^{(i)}jk$ with a relatively low absolute value implies that the j-th input of $\mathbf{x}^{(i)}$ is less important to the k-th output $y^{(k)}$ of layer i. Therefore, $W^{(i)}jk$ can be removed with minimal effect on the model's performance.

- After forward propagation for a dataset, If a set of neurons $\mathcal{E}^{(i)}j$ in layer i has low entropy, then those neurons contribute less to the model's prediction. By

averaging the entropy of all neurons affected by a certain parameter $W^{(i)}jk$, the importance score of $W^{(i)}jk$ can be inferred.

Based on this intuition, NEP fuses neuron entropy with model weight to obtain a comprehensive measure of parameter importance. However, for most linear affine transformations, the shapes of neurons and parameters differ. Neuron entropy measures the uncertainty of output and has the same shape as the layer output, while the parameters transform the layer input $\mathbf{x}^{(i)}$ to the pre-activation $z^{(i)}$ and can be viewed as a matrix.

To address this gap, NEP leverages the neuron entropy of all neurons in layer i to compute an importance score for each parameter. Specifically, given a parameter $W_j^{(i)}$, it averages the entropy of all neurons affected by this parameter as the overall entropy for this parameter. Thus, an importance score for the weight can be computed, having the same shape as the weights. Section 5.4.2.1 and Section 5.4.2.2 describe how this process is applied to linear and convolutional layers.

Essentially, the computation of the posterior importance score aims to extract insights into the neural network's expressive ability from neuron-level performance after forward propagation. To properly remove unnecessary neurons, the model weights, referred to as the prior importance score $\mathbf{IS_{prior}}$, also need to be considered. Section 5.4.2.3 introduces several methods to fuse these metrics to enhance model expressive ability.

5.4.2.1 Linear Layer

Let the i-th block of neural network $\mathcal{N}$ be a linear block. It takes $\mathbf{x}^{(i)}$ as input and computes an output $z^{(i)}$, where $\mathbf{x}^{(i)}$ and $z^{(i)}$ are vectors of size n_i and n_{i+1}, respectively. The weight matrix $W^{(i)}$ of block i has a shape of $n_i \times n_{i+1}$. This linear transformation can be described as:

$$z_j^{(i)} = \sum_{k=1}^{n_i} W_{jk}^{(i)} \mathbf{x}_k^{(i)}. \tag{5.25}$$

As $z^{(i)}$ is the pre-activation of layer i, neuron entropy can be computed for each

neuron $z_k^{(i)}$. The neuron $z^{(i)}k$ is influenced by the inputs $\mathbf{x}^{(i)}$ and the k-th row of the weight matrix $W^{(i)}$, as described above. This implies that the entropy of neuron (i, k) reflects the importance of the weights $W^{(i)}k, :$.

Based on this discussion, given a linear transformation, NEP computes the entropy for each output dimension as an overall entropy for the i-th row of the weight matrix $W^{([)}i, :]$. Intuitively, if the entropy of neuron (i, k) is low, then this neuron is firing the same signals regardless of the input, thereby losing expressive ability. The posterior importance score for the weight matrix is denoted as:

$$\mathbf{IS}_{post}(W^{(i)})_{k,:} = \frac{1}{n_i} \sum_{k=1}^{n_i} \mathcal{E}_k^{(i)}, \tag{5.26}$$

where $W_{k,:}^{(i)}$ is the k-th row of the weight matrix and $\mathcal{E}_k^{(i)}$ is the entropy of k-th neuron in layer i.

5.4.2.2 Convolutional Layer

Similar to that of a linear layer, the computation of the posterior importance score of a convolutional block also averages the entropy of all the neurons affected by the parameter. Denote $W^{(i)}$ as the weight of a convolutional layer. Assume that the shape of $W^{(i)}$ is $n_{in} \times n_{out} \times n_{k1} \times n_{k2}$, where n_{in}, n_{out}, n_{k1}, and n_{k2} are the number of input channels, the number of output channels, and the two kernel sizes, respectively. The input $\mathbf{x}^i$ and output z^i are the input and output of layer i, with shapes $n^i in \times n^{i-1} l \times n^{i-1} w$ and $n^i out \times n^i l \times n^i w$, where $n^{i-1} l$, $n^{i-1} w$, $n^i l$, and $n^i w$ are the lengths and widths of the layer's input and output. The discrete convolutional layer computes:

$$z_j^i = \sum_{k=0}^{C_{in}} W_{k,j,:,:}^i \star \mathbf{x}_k^i, \tag{5.27}$$

where $\mathbf{x}^i k$ is the k-th input channel, $z^i j$ is the j-th output channel, $W_{k,j,:,:}^i$ is the 2-D slice of the weight from the k-th input to the j-th output channel, and $\star$ is the 2D cross-correlation operator.

As the neuron entropy of layer i is observed and computed from the output $z^i j$, the importance score for the weights should take the output $z^i j$ into account. It can

be noticed that the j-th filter of the output is computed from the j-th slice cut from the output dimension of the weight, which means that each parameter in $W^i k, j, :, :$ contributes to the prediction of the j-th channel of the output. Following the same idea of computing the importance score of a parameter using the entropy of the neurons affected by it, the posterior importance score of filter $W^i j, :, :, :$ is computed by averaging the entropy of the output. This can be represented as:

$$\mathbf{IS}_{post}(W)[j, , :, :] = \frac{1}{n_l^i} \frac{1}{n_w^i} \sum_{a=0}^{n_l^i} \sum_{b=0}^{n_w^i} \mathcal{E}_{j,a,b}^{(i)}, \tag{5.28}$$

where $\mathcal{E}_{j,a,b}^{(i)}$ is the neuron entropy of (a, b)-th output in channel j of layer i.

5.4.2.3 Blending Methods

After preparing the weights and neuron entropy, the next step is to blend the prior and posterior measures of parameter importance. As suggested by their names, the prior importance score measures the usefulness of a parameter before feeding any data, based on the observation that parameters with lower absolute values have less effect on prediction. On the other hand, the posterior importance score is computed from neuron entropy and measures expressive ability based on the reaction of neurons to a dataset.

The computation of the posterior importance score provides insights into the neural network's expressive ability from neuron-level performance after forward propagation. To properly remove unnecessary neurons, the weights of the model—referred to as the prior importance score—must also be considered. This blending of metrics involves balancing the prior and posterior measures of expressive ability. In the following experiments, this section considers the following approaches:

- Weight:

$$\mathbf{IS}(W^i) = \mathbf{IS}(W^i)$$

- Entropy weighted:

$$\mathbf{IS}(W^i) = \mathbf{IS}_{post}(W^i)\mathbf{IS}_{prior}(W^i).$$

- Harmonic mean of weight and layer entropy:

$$\mathbf{IS}(W^i) = \frac{1}{\frac{1}{\mathbf{IS}_{post}(W^i)} + \frac{1}{\mathbf{IS}_{prior}(W^i)}}.$$

- Normalized weight × Normalized entropy:

$$\mathbf{IS}(W^i) = \frac{\mathbf{IS}_{pre}(W^i) - \mu_{pre}}{\sigma_{pre}} \times \frac{\mathbf{IS}_{post}(W^i) - \mu_{post}}{\sigma_{post}},$$

  where μ_{pre} and μ_{post} are the mean of $\mathbf{IS}_{pre}$ and $\mathbf{IS}_{post}$, σ_{pre} and σ_{post} are the variance of $\mathbf{IS}_{pre}$ and $\mathbf{IS}_{post}$.

- Standardized weight × Standardized entropy:

$$\begin{aligned}\mathbf{IS}(W^i) = &\frac{\mathbf{IS}_{pre}(W^i) - \min(\mathbf{IS}_{pre}(W^i))}{\max(\mathbf{IS}_{pre}(W^i)) - \min(\mathbf{IS}_{pre}(W^i))} \times \\ &\frac{\mathbf{IS}_{post}(W^i) - \min(\mathbf{IS}_{post}(W^i))}{\max(\mathbf{IS}_{post}(W^i)) - \min(\mathbf{IS}_{post}(W^i))}.\end{aligned} \tag{5.29}$$

The first method uses only weights as the importance score, serving as a benchmark for the pruning experiments to illustrate that introducing neuron entropy is helpful in identifying unimportant parameters. The entropy-weighted and harmonic mean methods combine the prior and posterior importance scores in different ways. The normalized weighted and normalized summed methods apply normalization to the values to remove the magnitude difference between the two measures.

5.4.3 Neuron Entropy Pruning

Now that the methods of computing the parameter importance score have been introduced, the remaining question is how the algorithm is applied to a neural network.

Algorithm 1 presents the detailed training and pruning steps of NEP. At the beginning of the training, a mask M is initialized with the same shape as the network parameters to denote the remaining neurons. During the training, both forward and backward propagation are performed on the masked parameters, as suggested in

Algorithm 1: NEP training algorithm

Input: Training data $\mathcal{D}_{\text{train}}$, validation data $\mathcal{D}_{\text{val}}$, number of epochs T, pruning milestones P, breakpoints Γ, pruning threshold e_b

Function `Train`$(f, \boldsymbol{\theta})$ **:**
  Initialize Mask M with 1.
  for *Epoch* $t = 1 \dots T$ **do**
    while $(\mathbf{x}, y) \in D_{train}$ **do**
      $\hat{y} = f(\mathbf{x}|M \cdot \boldsymbol{\theta})$
      $\boldsymbol{\theta} \leftarrow M \cdot \boldsymbol{\theta} - M \cdot \frac{\partial \text{loss}(\hat{y}, y)}{\partial \boldsymbol{\theta}}$
    end
    if $t \in P$ **then**
      $M \leftarrow$ `Prune`$(\boldsymbol{\theta}, t)$
    end
  end
  return $\boldsymbol{\theta}$
Function `Prune`$(\boldsymbol{\theta}, t)$ **:**
  Initialize activation pattern counter pc.
  while $(\mathbf{x}, y) \in \mathcal{D}_{train}$ **do**
    for *Block* $i = 1 \dots d$ **do**
      $\mathbf{z} \leftarrow \psi_i \circ \phi_i(\mathbf{x})$
      $\hat{a}_j^{(i)}(\mathbf{x}) \leftarrow$ `ComputePattern`$(\mathbf{z}, \Gamma)$
      $pc_j^{(i)}[\hat{a}_j^{(i)}(\mathbf{x})] \leftarrow pc_j^{(i)}[\hat{a}_j^{(i)}(\mathbf{x})]$
      $\mathbf{x} \leftarrow \sigma_i(\mathbf{z})$
    end
  end
  $pc_j^{(i)} \leftarrow pc_j^{(i)} / \#\mathcal{D}_{\text{train}}$
  $\mathcal{E}_{ij} \leftarrow$ `Entropy`$(pc_j^{(i)})$
  $M_j^{(i)} \leftarrow M_j^{(i)} \wedge$ `PruneCriteria`$(\mathcal{E}_j^{(i)})$
  return M

steps 5 and 6. Step 9 prunes the network if the current epoch is within the pruning milestone P.

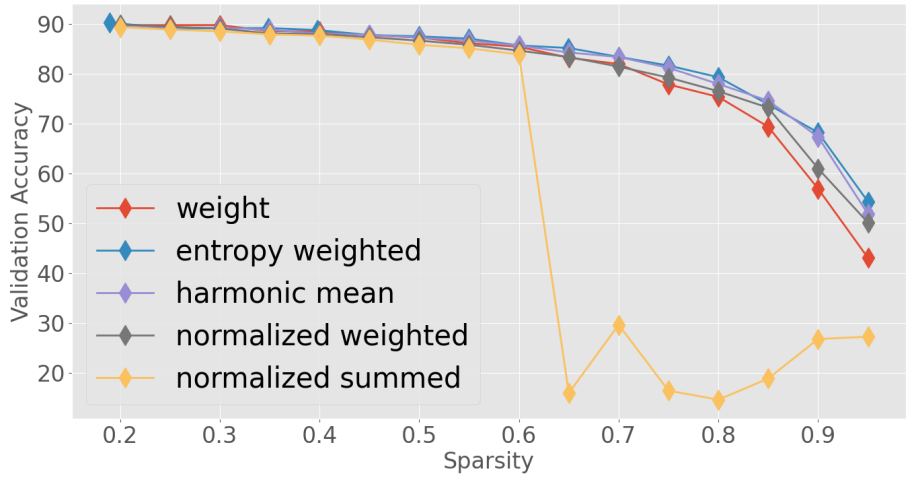
The prune function computes the entropy of each neuron and applies a mask to the neurons with the lowest entropy. A counter pc is first initialized, where $pc^{(i)}j[k]$ is the number of occurrences of pattern k for neuron (i, j). For each block h_i, the pre-activation z_i is first computed at step 17, followed by the activation pattern for each neuron $\hat{a}^{(i)}j$ at step 18. The pattern counter of the current pattern $\hat{a}^{(i)}j$ for neuron (i, j) is updated by 1. After passing through the entire dataset, the frequency of each pattern can be viewed as an estimation of the pattern distribution for neuron (i, j). We then compute the entropy of neuron (i, j) as $\mathcal{E}^{(i)}j$ using $pc^{(i)}j$. After processing the entire dataset, all neurons that meet the pruning criteria are pruned by setting the mask $M^{(i)}j = 0$.

The network starts with a standard training process. After a certain number of epochs, the neuron entropy of the network on the validation set is computed, and a mask is applied to the neurons with the lowest entropy. We separate the training and pruning steps in Algorithm 1 for clarity. Empirically, the computation of neuron entropy can be merged into the training steps, and the update of the pattern counter can be performed batch-wise.

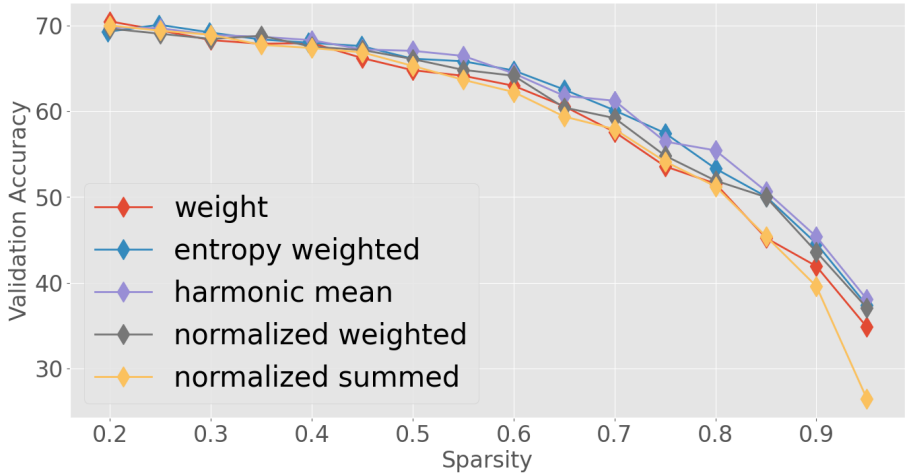
5.4.4 Pruning

This section presents the experiment results of NEP. The algorithm first evaluates the expressive ability of the neural network using neuron entropy metrics. Based on the volatility of neurons, it is shown that a certain number of *dying neurons* fail to contribute to the network's prediction. The neuron entropy pruning method aims to leverage this information by combining the weights and neuron entropy to eliminate unnecessary parameters from the network.

We begin by investigating the performance of networks pruned using different methods. Each network is initialized with He Normal Distribution [18] and trained using the SGD optimizer on the CIFAR10 dataset with a batch size of 128 for 120



(a) VGG16



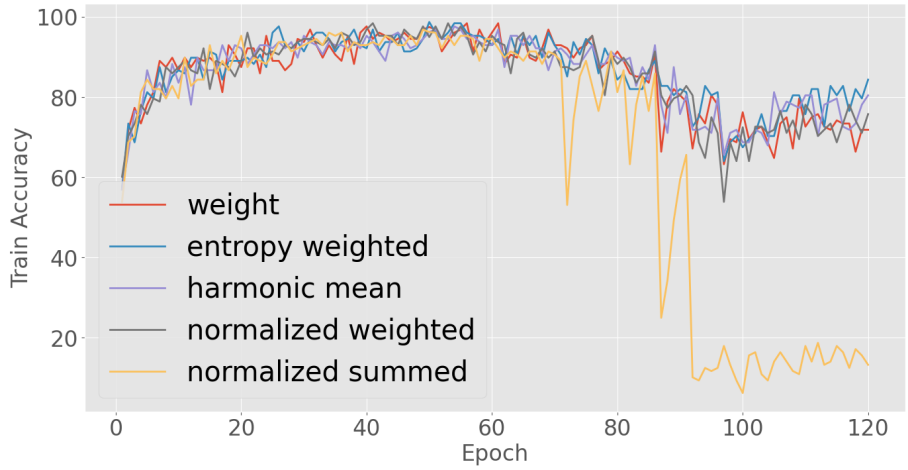
(b) AlexNet

Figure 5.11: Comparison between Pruning Methods with different sparsity

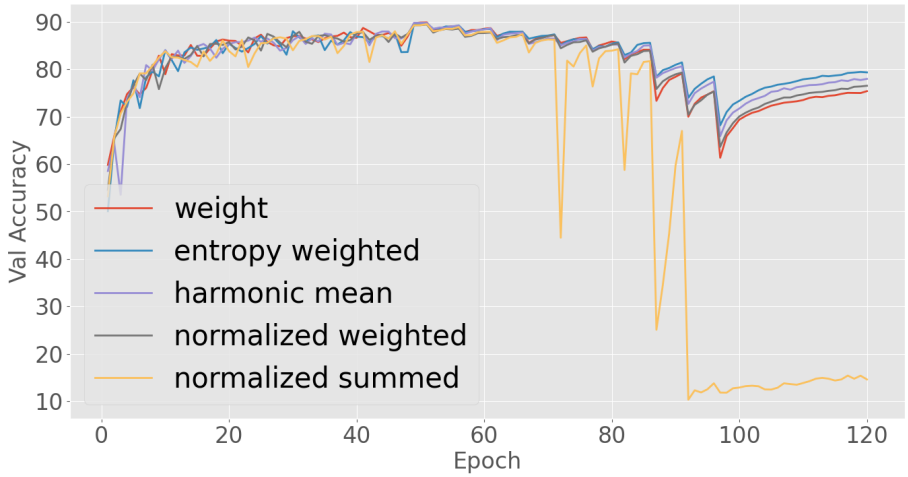
epochs. The initial learning rate of 0.1 decays by a factor of 10 after 48 and 84 epochs. To prevent the update of pruned parameters, we set the weight decay and momentum of the optimizer to 0.

5.4.4.1 Pruning Details

In this set of experiments, we prune the network every 10 epochs to allow the network to "recover" and optimize the remaining parameters. This training procedure is justified in Section 5.4.4.3. Near the end of training, we stop pruning and fine-tune



(a) Train Accuracy



(b) Validation Accuracy

Figure 5.12: Comparison between Pruning Methods with different sparsity

the network for 20 epochs. At each pruning epoch, the entropy of each neuron is recorded during training, and an importance score is computed at the end of training by blending the layer entropy with the model weight. Parameters from different layers are removed in equal proportion according to the importance score.

5.4.4.2 Comparison between Pruning Methods

Figure 5.11 compares the accuracy of networks pruned using different blends. Compared with the benchmark model (Weight), most of the models pruned by an

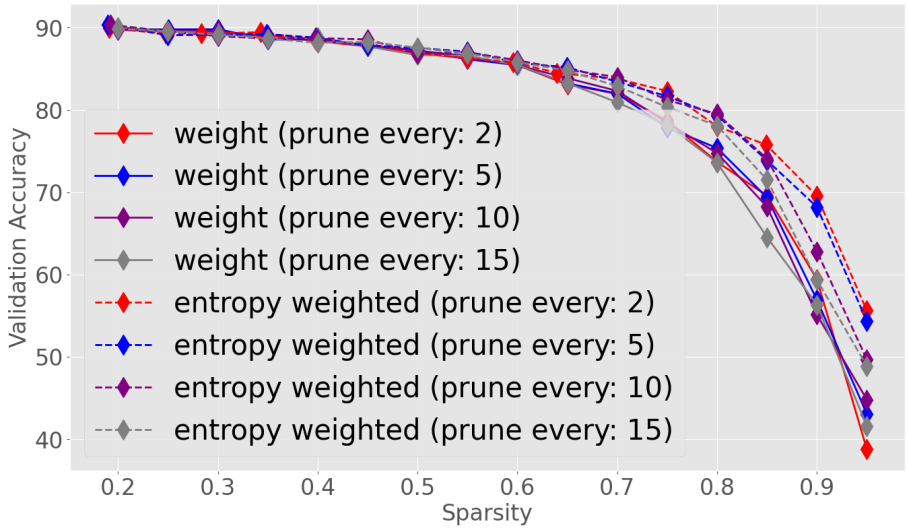


Figure 5.13: Comparison of Prune Frequency with different sparsity

importance score blended with weights and entropy achieve higher accuracy. Among them, the model using entropy-weighted parameters achieves the best performance, boosting the accuracy of the benchmark model by 3% to 8% at various sparsity levels from 0.7 to 0.95, followed by *harmonic mean* and *normalized weighting*. This suggests that more useful parameters survive when neuron entropy is considered during the pruning process. In other words, neuron entropy can be viewed as a metric for evaluating the expressive power of the network.

5.4.4.3 Pruning Frequency

To understand the effect of pruning on models during training, Figure 5.12 reports the training and validation accuracy of the pruning method with 80% sparsity. The model accuracy drops after each pruning and gradually recovers during the non-pruning epochs. As model sparsity increases, the performance impairment is greater, and the model finds it harder to recover. This suggests that the pruning frequency and fine-tuning steps can affect the final performance of the network.

Figure 5.13 compares model accuracy given different pruning frequencies under different target sparsity. Given sparsity S, models are pruned every 2, 5, 10, or 15

epochs with 20 fine-tuning epochs, resulting in 49, 19, 9, and 6 pruning iterations, respectively, where the same proportion of parameters is removed each time. Low pruning frequency implies a larger amount of pruning each time and a longer fine-tuning period.

For the models pruned by weights, increasing the pruning frequency results in slightly lower validation accuracy. The model with a pruning frequency of 10 outperforms the others by around 7%. On the other hand, when neuron entropy is introduced to the importance score, NEP (entropy-weighted) models with higher pruning frequency achieve better results. Increasing the pruning frequency from 15 to 2 boosts model performance by around 6%. In general, considering pruning frequency, the proposed NEP (entropy-weighted) outperforms the benchmark models by around 12%.

5.5 Chapter Summary

This chapter investigates the expressive ability of neural networks with respect to the *dying neuron issue*. The dying neuron issue refers to the phenomenon where most neurons produce similar outputs for any data, resulting in a loss of practical expressive ability. Unlike the gradient vanishing issue, the gradient of a *dying neuron* is non-zero. However, because most of these neurons fire similar outputs regardless of the input, they fail to provide useful information, leading to a reduced expressive ability.

The theoretical results of this chapter are presented in Sections 5.1 and 5.3. In particular, two metrics—*pattern similarity* and *neuron entropy*—are introduced to measure the expressive ability of neural networks, focusing on:

- Exploring the expressive ability across models with different activation functions. This analysis provides an explanation for the introductory question and shows that Sigmoid nets and Tanh nets suffer more severely from the *dying neuron issue* than other models.

- Investigating the *dying neuron* issue within a single model, offering insights into the roles of different layers in function approximation.

Pattern similarity evaluates the model's expressive ability across different models. The experiments in Chapter 5.2 establish a link between pattern similarity and model performance. They explain that the poor performance of Sigmoid nets can be attributed to the *dying neuron* issue, where most neurons fire similar signals toward data with different labels, thus failing to distinguish their differences.

On the other hand, neuron entropy evaluates neuron-wise volatility and aims to understand why models lose generalization ability on out-of-sample datasets. It was found that in deep layers, neurons tend to lose their volatility and suffer from the dying neuron issue, meaning they do not contribute to the prediction. Inspired by this, Chapter 5.4 introduces a model pruning method that computes the importance score of neurons using both weights and neuron entropy. The experiments in 5.4 show that, compared to benchmark models, the proposed method can reduce the model's scale while maintaining better accuracy on validation datasets.

Chapter 6

Float Path and Model Robustness

As another mainstream research topic, a significant amount of research has explained how perturbations affect the prediction results of neural networks from different perspectives. This chapter discusses the robustness of neural networks within the proposed framework.

In Chapter 6.1, the concepts of float paths and fixed paths were introduced to describe the stability of neurons within a region. Building on this framework, the investigation in this chapter is based on the idea of connecting the instability of neural networks to neuron behavior. Although similar bounds and insights can be derived for non-piecewise linear functions, this chapter focuses on neural networks with *piecewise linear activation functions* for a clearer understanding.

According to Definition 8, given a subspace $\mathcal{R} \subseteq \mathbb{R}^n$, the activation pattern of every neuron along a path ζ remains unchanged if ζ is a fixed path in $\mathcal{R}$. This implies that the mapping from the input space to the output space of $\zeta(\mathbf{x})$ remains consistent. Conversely, the mapping along float paths changes within the region. Since the computational graph of a feedforward neural network is a directed acyclic graph, variations in prediction can be attributed to changes in paths if the computational graph can be decomposed into such paths.

Based on this idea, Chapter 6.2 describes a method for decomposing the neural network. It is shown that the network can be decomposed into a fixed path $\mathcal{Z}^I$

and a float path $\mathcal{Z}^T$ within the region $\mathcal{R}$. The unsecured data within $\mathcal{R}$ is further categorized into *float vulnerable* and *Lipschitz vulnerable* according to which part of the decomposition affects it. Chapter 6.3 then investigates the effects of robustness training methods. It is found that adversarial training methods can compress the Lipschitz constant of the network, addressing the Lipschitz vulnerability, but they do not effectively address the float vulnerability. On the other hand, the smoothed classifier can provide robust predictions for unsecured data by averaging the float paths $\mathcal{Z}^T$.

To improve model robustness, Chapter 6.4 introduces a method named Smoothed Classifier with Refactored Float Paths in Dual Directions (SCRFP-2). By manipulating the behavior of float neurons during training and prediction, this method further improves the performance of the smoothed classifier. To provide a theoretical bound for the method, it is shown that the certifiable radius of the proposed method holds as long as the randomness generated by the smoothed classifier is directionally irrelevant. The experimental results and comparisons with previous works are then presented in Chapter 6.5.

6.1 Activation Paths

In the previous section, a framework was presented to describe the neural network within an arbitrary subspace $\mathcal{R} \subset \mathbb{R}^n$, along with a general discussion of the upper bound of the network. Although the result is insightful for understanding the performance of a neural network, a more precise discussion of network prediction is also preferred.

The next step in building the theoretical framework is to make use of the above definitions by connecting the mapping relationship of the network with the status of neurons. The idea of this section is to decompose the computational graph of the neural network, enabling an in-depth analysis of how prediction variation is caused by changes in activation status. Due to the high complexity of neural networks, this

decomposition is performed on networks with *piecewise linear* activation functions.

This section introduces definitions related to the computational graph of the neural network. A feedforward neural network can be represented as a directed acyclic graph. This means that the computational graph of such a network can be decomposed into a sum of paths, where each path starts from the inputs, connects through a unique neuron at each layer, and terminates at the network output. To investigate the expressive ability and robustness of a neural network, the framework proposed in this research decomposes the computational graph of the network into paths. A path of a feedforward neural network $\mathcal{N}$ is defined as:

Definition 7 (Activation Path)**.** *Let $\mathcal{N}$ be a neural network as defined in Section 4.1. A path of $\mathcal{N}$ is a set of neurons that:*

$$\zeta := \{\zeta_i | i = 0, 1, \ldots, n\} \in Z_{R_0} \times Z_{R_1} \cdots \times Z_{R_n} \tag{6.1}$$

where $Z_{R_i} = \{1, 2, \ldots, n_i\}$ is neuron indices for layer i. A sub-path of a path consists of several consecutive elements of γ:

$$\zeta_{(i,j)} = \{\zeta_j, \zeta_{j+1} \ldots, \zeta_i\} \subseteq \zeta \tag{6.2}$$

A trivial insight is that the value of each path is determined by the input $\mathbf{x} \in \mathbb{R}^n$ as well as the status of the neurons along this path. This means that given a subspace $\mathcal{R} \in \mathbb{R}^n$, if all the neurons in a path ζ have the same activation status, ζ provides a stable contribution to the network's prediction; otherwise, ζ is unstable. We use the terms float and fixed neuron/path to describe this:

Definition 8 (Float Path and Fixed Path)**.** *Let $\mathcal{N}$ be a neural network defined as in Section 4.1. Given a subspace $\mathcal{R} \subset \mathbb{R}^n$, a path ζ of $\mathcal{N}$ is a float path in $\mathcal{R}$ if there exists at least one neuron $\zeta_i \in \zeta$ that is a float neuron in $\mathcal{R}$; otherwise, the path ζ is a fixed path.*

$$\begin{aligned} &\textit{float path in } \mathcal{R} := \{\zeta | \exists \zeta_i \in \zeta, (i, \zeta_i) \notin \mathcal{I}^I(\mathcal{R})\} \\ &\textit{fixed path in } \mathcal{R} := \{\zeta | \forall \zeta_i \in \zeta, (i, \zeta_i) \in \mathcal{I}^I(\mathcal{R})\}. \end{aligned} \tag{6.3}$$

The value of float paths and float paths in R are denoted as:

$$\mathcal{Z}^T(\mathbf{x}, \mathcal{A}; R) = \sum_{\textit{float path in } \mathcal{R}} \zeta(\mathbf{x}, \mathcal{A}),$$
$$\mathcal{Z}^I(\mathbf{x}, \mathcal{A}; R) = \sum_{\textit{fixed path in } \mathcal{R}} \zeta(\mathbf{x}, \mathcal{A}),$$

where $\zeta(\mathbf{x}, \mathcal{A})$ is the value of path ζ given input x and an activation pattern $\mathcal{A}$.

Similar to the float/fixed neurons, the float/fixed paths are also defined on a subspace $\mathcal{R} \in \mathbb{R}^n$. For any $\mathbf{x}, \mathbf{x}' \in \mathcal{R}$, fixed neurons have the same activation status. If all the neurons on a path are fixed neurons, then ζ is a fixed path, which means the value of ζ is stable in the region $\mathcal{R}$. In particular, if the activation π of the network is a piecewise linear function, then the value of ζ is linear in the region $\mathcal{R}$. On the contrary, a float path in the region $\mathcal{R}$ introduces more non-linearity to the network's prediction. Therefore, the above definition decomposes the prediction $f(\mathbf{x})$ of a network according to the stability of the computational paths, where $Z^I(\mathbf{x}, \mathcal{A}; \mathcal{R})$ and $Z^T(\mathbf{x}, \mathcal{A}; \mathcal{R})$ are the values of fixed paths and float paths, respectively.

In the following, the decomposition of the computational graph of deep neural networks is further discussed to investigate the robustness of neural networks. As defined in Definition 4, the unstable neurons for a given region $\mathcal{R}$ can be identified by the framework proposed in this research, making the decomposition helpful in understanding the robustness of neural networks. The discussion in this dissertation is limited to models with *piecewise linear* activation functions due to the high complexity of networks with deeper structures. For non-piecewise linear activation functions, a similar bound can also be obtained by following the ideas in Lemma 11 and Theorem 8, which will be discussed in future works.

6.2 Decomposing the Computational Graph

This section discusses the robustness of a neural network within the proposed framework. The research of model robustness focuses on enhancing the reliability

and stability of models under a variety of conditions. In deep learning, given a well trained model N, it is found that some data x can be deliberately modified, which is denoted as x', in a way that is often imperceptible to humans but causes the model N deep learning model to make mistake. Then x' is known as an adversarial example, while the original input x is an unsecured data.

The objective of this section is to provide insights on understanding the unsecured data from actiavtion paths perspective. It begins by describing how float and fixed paths contribute to the prediction difference between $f(\mathbf{x})$ and $f(\mathbf{x}')$ given $\mathbf{x} \in B(\mathbf{x}, r)$. Based on this discussion, unsecured data is then categorized into Lipschitz vulnerable and float neuron vulnerable types, and the two types of vulnerability are investigated. Given a trained model N and an adversarial attack, if there exists an adversarial example x', then x is an unsecured data. This section then investigates the behaviour of float paths and fixed paths of those unsecured data from themodel N.

To understand the motivation, consider the simple case where there is only one neuron (i, j) that is a float neuron in the region $R \subset \mathbb{R}^{n_0}$. The only non-linearity in $\mathcal{R}$ is provided by neuron (i, j). The computational graph of $\mathcal{N}$ can be decomposed into *float paths* and *fixed paths* based on whether (i, j) is on the path. This is formally described in the first statement of Theorem 8. Furthermore, given $\mathbf{x}, \mathbf{x}' \in \mathcal{R}$, the difference between $f(\mathbf{x})$ and $f(\mathbf{x}')$ can be viewed as a combination of a linear function and an unstable function with high non-linearity, as suggested by the second statement in Theorem 8. The remaining challenge is how to decompose the computational graph of network $\mathcal{N}$, which is discussed in the following section.

6.2.1 Decomposing the Computational Graph

The framework for describing the status of neurons is presented in Chapter 4.3. The following lemma establishes a connection between the mapping function f and the activation pattern given a piecewise linear activation function π.

Lemma 11 (Decomposition of Neural Network). *Let $\mathcal{N}$ be a neural network defined as in Chapter 4.1 with a piecewise linear activation function π. Given activation pattern $\mathcal{A}$, for any $x \in \mathcal{R}(\mathcal{A}; \theta, \sigma, \Gamma)$, the k-th component of the output of layer i can be represented as:*

$$y_k^{(i)}(\mathbf{x}) = \sum_{\forall \zeta_i = k} \mathbf{x}_{\zeta_j}^{(j)} \prod_{m=j+1}^{i} d_{\mathcal{A}_{m,\zeta_m}} W^{(m)}_{(\zeta_m, \zeta_{m-1})} + \sum_{o=j}^{i-1} \sum_{\forall \zeta_i = k} \beta_{\zeta_o}^{(o)} \prod_{m=o+1}^{i} d_{\mathcal{A}_{m,\zeta_m}} W^{(m)}_{(\zeta_m, \zeta_{m-1})} \tag{6.4}$$

where $\mathbf{x}_{\zeta_j}^{(j)}$ is the ζ_j-th element of input in layer j, $W'^{(i)}$ and $\beta^{(i)}$ are the equivalent matrix and bias of linear transformation $\psi_i \circ \phi_i$, $\mathcal{A}_{m,\zeta_m}$ is the activation pattern of ζ_m-th component of layer m and $d_{\mathcal{A}_{m,\zeta_m}}$ is the slope of pattern $\mathcal{A}_{m,\zeta_m}$:

$$d_{\mathcal{A}_{m,\zeta_m}} := \sigma'(t), t \in U_{\mathcal{A}_{m,\zeta_m}}.$$

Proof (Lemma 11). *We first show that the pre-activation transformation can be represented by a matrix. As ϕ_i is a linear affine, we denote $\phi_i(x) = W_i x_i$. Combing with batch normalization layer, the mapping from input x_i to pre-activation is:*

$$z_i = \gamma_i \frac{\phi_i(x) - \hat{\mu}}{\hat{\sigma}} + \beta_i = \frac{\gamma}{\hat{\sigma}} \phi_i(x) - \frac{\gamma \hat{\mu}}{\hat{\sigma}} + \beta_i = \mathbf{Diag}_i(\frac{\gamma}{\hat{\sigma}}) W_i x - \frac{\gamma \hat{\mu}}{\hat{\sigma}} + \beta_i \tag{6.5}$$

where $\mathbf{Diag}_i(\cdot)$ is the diagonal operator, $\hat{\mu}$ and $\hat{\sigma}$ are the mean and variance parameter of the ψ_i. Denote $W'^{(i)} = \mathbf{Diag}_i(\frac{\gamma}{\hat{\sigma}}) W_i$, $\beta_i' = (\beta_i - \frac{\gamma \hat{\mu}}{\hat{\sigma}})/n_i$. This is proved by deduction. For $i = 1$:

$$z_i = \sum_{j=0}^{n_0} W_{ij}^{(1)} d_{\mathcal{A}_{1,i}} x_{0j} + n_1 \times \beta_i'^{(1)},$$

where $\mathcal{A}_{1,i}$ is the activation pattern of z_{1i}. Since all the path in layer i ends at i are $\{(1,i), (2,i), \ldots, (n_0, i)\}$. Equation 6.4 holds.

Assume Equation 6.4 holds for $i = p$.

$$
\begin{aligned}
z_{pi} &= \sum_{j=0}^{n_i} W'^{(i)} x_{ij} + n_i \times \beta_j'^{(i)} \\
&= \sum_{j=0}^{n_i} W'^{(i)} z_{i-1,j} + n_i \times \beta_j'^{(i)} \\
&= \sum_{j=0}^{n_i} W'^{(i)} d_{\mathcal{A}_{i,j}} z_{i-1,j} + n_i \times \beta_j'^{(i)} \\
&= \sum_{j=0}^{n_i} W'^{(i)} d_{\mathcal{A}_i,j} (\sum_{\forall \zeta_i = j} x_{j,\zeta_j} \prod_{m=p}^{i} d_{\mathcal{A}_{m,\zeta_m}} W'^{(m)}_{(\zeta_m,\zeta_{m-1})} \\
&+ \sum_{o=p-1}^{i-1} \sum_{\forall \zeta_i = k} \beta^{(o)}_{\zeta_o} \prod_{m=o+1}^{i} d_{\mathcal{A}_{m,\zeta_m}} W'^{(m)}_{(\zeta_m,\zeta_{m-1})}) + \beta_j'^{(i)} \\
&= \sum_{\forall \zeta_p = k} x_{j,\zeta_j} \prod_{m=j+1}^{p} d_{\mathcal{A}_{m,\zeta_m}} W'^{(m)}_{(\zeta_m,\zeta_{m-1})} + \sum_{o=j}^{p-1} \sum_{\forall \zeta_p = k} \beta^{(o)}_{\zeta_o} \prod_{m=o+1}^{i} d_{\mathcal{A}_{m,\zeta_m}} W'^{(m)}_{(\zeta_m,\zeta_{m-1})}
\end{aligned}
\tag{6.6}
$$

Lemma 11 decomposes f in terms of input and activation pattern. Equation 6.4 shows that $y^{(i)}j(\mathbf{x})$ can be decomposed into a summation of paths, where the value of each path depends on either the input $\mathbf{x}$ (former part) or the bias β at each layer (latter part), as well as the pattern of neurons on the path. Given v as the input of a path, denote the value of the path as $\zeta(i,j)(v,\mathcal{A})$:

$$
\zeta_{(i,j)}(v,\mathcal{A}) := v \prod_{m=j+1}^{i} d_{\mathcal{A}_{m,\zeta_m}} W^{(m)}_{(\zeta_m,\zeta_{m-1})}. \tag{6.7}
$$

In a single activation region $\mathcal{R}$, $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{R}$ have the same activation pattern $\mathcal{A}$. The paths from bias $\zeta_{(i,j)}(\beta^{(j)}_{\zeta_j}, \mathcal{A})$ are constant, while those from input $\zeta_{(i,j)}(\mathbf{x}^{(j)}\zeta_j, \mathcal{A})$ are linear with respect to xj, ζ_j. On the other hand, for $\mathbf{x}, \mathbf{x}'$ with different activation patterns $\mathcal{A}, \mathcal{A}'$, each path can be categorized as either fixed or float. Since neurons on fixed paths have the same pattern at $\mathbf{x}$ and $\mathbf{x}'$, the above linearity remains unchanged, which means $\zeta_{(i,j)}(v,\mathcal{A}) = \zeta_{(i,j)}(v,\mathcal{A}')$ if ζ is a fixed path. In other words, given region $R \subset \mathbb{R}^{n_0}$, the float paths represent the linear part of f in R, while the fixed paths are the aggregation of all the non-linearity from the float neurons, as described below:

Theorem 8. *Let N be a neural network defined as in Section 4. Given $R \subset X$, for any $x, x' \in R$ with activation patterns $\mathcal{A}$ and $\mathcal{A}'$, we have:*

1. $f(x) = \mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}) + \mathcal{Z}^T(x, \mathcal{A}; \mathcal{R})$
2. $f(x) - f(x') = J(x)(x - x') + \mathcal{Z}^T(x', \mathcal{A}; \mathcal{R}) - \mathcal{Z}^T(x', \mathcal{A}'; \mathcal{R})$

where, $\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}) = \sum_{\zeta \in \mathcal{Z}^I(\mathcal{R})} \zeta(x, \mathcal{A}), \mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}) = \sum_{\zeta \in \mathcal{Z}^T(\mathcal{R})} \zeta(x, \mathcal{A})$ are the sum of fixed path and float path given the region $\mathcal{R}$, $J(x)$ is the Jacobian matrix of f at x.

Based on Lemma 11, Theorem 8 decomposes the computational graph of the neural network $\mathcal{N}$ into fixed parts $\mathcal{Z}^I$ and float parts $\mathcal{Z}^T$. It further shows how each of these parts contributes to the difference $\delta(\mathbf{x}, \mathbf{x}')$ between $f(\mathbf{x})$ and $f(\mathbf{x}')$. Before presenting the proof of Theorem 8, the following lemma is introduced to show that every fixed path in a region is linear regardless of the activation pattern.

Lemma 12. *Let $\mathcal{N}$ be a network defined as Chapter 4.1. Let ζ be a fixed path in $R \subset R_{n_0}$. Then for any $x, x' \in R$ with activation pattern $\mathcal{A}, \mathcal{A}'$, $\zeta(x', \mathcal{A}) = \zeta(x', \mathcal{A}')$. Moreover, $\mathcal{Z}^I(x', \mathcal{A}; \mathcal{R}) = \mathcal{Z}^I(x', \mathcal{A}'; \mathcal{R})$.*

Proof (Lemma 12)**.** *Given $x \in R$ with activation pattern A,*

$$\zeta(x, \mathcal{A}) := v \prod_{m=1}^{d} d_{\mathcal{A}_{m,\zeta_m}} W'^{(m)}_{(\zeta_m, \zeta_{m-1})}.$$

Since ζ is a fixed path in R, then every neuron alone ζ is fixed neuron, the activation pattern of (m, ζ_m) is same regardless of input x. Therefore, $d_{\mathcal{A}_{m,\zeta_m}}$ is constant for any $x \in R$. Then $\zeta(x, \mathcal{A})$ is a linear function of $x : \zeta(x, \mathcal{A}) = \zeta(x)$. In other words, the change of activation pattern does not affect the neurons on ζ, therefore the slope of this path does not change. $\zeta(x)$ is dependent on x in region R. This means that $\zeta(x', \mathcal{A}) = \zeta(x', \mathcal{A}')$ for any $x, x' \in R$ with activation pattern $\mathcal{A}, \mathcal{A}'$

$\mathcal{Z}^I(x', \mathcal{A}; \mathcal{R})$ is the aggregation of all the fixed path above. Since the summation of linear function is still linear, we have:

$$\mathcal{Z}^I(x', \mathcal{A}; \mathcal{R}) = \mathcal{Z}^I(x', \mathcal{A}'; \mathcal{R})$$

Lemma 12 shows that, given two activation patterns $\mathcal{A}$ and $\mathcal{A}'$, if the corresponding activation regions $\mathcal{R}(\mathcal{A})$ and $\mathcal{R}(\mathcal{A}')$ are within the same region $\mathcal{R}(\mathcal{A}), \mathcal{R}(\mathcal{A}') \subset \mathcal{R}$, then the value of the fixed paths $\zeta(\mathbf{x}', \mathcal{A}) = \zeta(\mathbf{x}', \mathcal{A}')$. Since every neuron on the fixed path has the same pattern for any $\mathbf{x} \in \mathcal{R}$ and the value of $\zeta(\mathbf{x}', \mathcal{A})$ is only affected by the fixed neurons, the choice of activation pattern does not change the value of the path. With this, the proof of Theorem 8 is presented as follows.

Proof (Theorem 8)**.** *From Lemma 8, every neuron is either a fixed neuron or a float neuron. For the neurons in path ζ, if there exists a float neuron, then the path is a float path. Otherwise, it is a fixed path. The float paths and fixed paths form complementary sets within the set of all paths.*

Lemma 11 decomposes $f(\mathbf{x})$ into a summation of paths. Since each path is either float or fixed, the prediction of $f(\mathbf{x})$ can be written as:

$$f(x) = \mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}) + \mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}),$$

as described by the first statement. For the second statement, it can be shown that:

$$\begin{aligned} f(x) - f(x') &= \big(\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}) + \mathcal{Z}^T(x, \mathcal{A}; \mathcal{R})\big) - \big(\mathcal{Z}^I(x', \mathcal{A}'; \mathcal{R}) + \mathcal{Z}^T(x', \mathcal{A}'; \mathcal{R})\big) \\ &= \big(\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}) + \mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}) - \mathcal{Z}^I(x', \mathcal{A}; \mathcal{R}) - \mathcal{Z}^T(x', \mathcal{A}; \mathcal{R})\big) + \\ &\quad \mathcal{Z}^T(x', \mathcal{A}; \mathcal{R}) - \mathcal{Z}^T(x', \mathcal{A}; \mathcal{R}) \end{aligned} \tag{6.8}$$

Notice that, $\mathcal{Z}^I(x', \mathcal{A}; \mathcal{R})$ is the collection of all fixed path, therefore is unrelated with the change of $\mathcal{A}$. We have $\mathcal{Z}^I(x', \mathcal{A}; \mathcal{R}) = \mathcal{Z}^I(x', \mathcal{A}'; \mathcal{R})$. The former part of above equation is equals to $J(x)(x - x')$.

With Lemma 11, Theorem 8 decomposes the computational graph of network $\mathcal{N}$ into a linear part and a non-linear part. Given $\mathbf{x}$ and $\mathbf{x}'$, by subtracting the difference between their prediction results $f(\mathbf{x})$ and $f(\mathbf{x}')$ and rearranging the equation, the difference $\delta(\mathbf{x}, \mathbf{x}')$ can also be described in a similar form.

The second statement of Theorem 8 suggests that the difference between $f(\mathbf{x})$ and $f(\mathbf{x}')$ in network $\mathcal{N}$ can be decomposed into: (1) a linear part that depends

on the distance between $\mathbf{x}$ and $\mathbf{x}'$ as well as the slope at $\mathbf{x}$, and (2) a non-linear part that results from the instability of float neurons. According to the primary contributor to prediction differences, unsecured data can be categorized as either *Lipschitz vulnerable* or *float neuron vulnerable*. In the next section, the robustness of a neural network with respect to input perturbation is discussed.

6.3 Two Types of Vulnerabilities

Prior to delving into our analysis, we first provide an intuitive discussion of how float neurons affect model robustness. Given $\mathbf{x}$ and $\mathbf{x}' \in B(\mathbf{x}, r)$, the second statement in Theorem 8 suggests that the difference between $f(\mathbf{x})$ and $f(\mathbf{x}')$ can be decomposed into fixed paths and float paths. The fixed paths are locally linear, while the float paths can result in sudden changes between the predictions. Therefore, a stable network should have a lower Lipschitz constant as well as fewer float neurons.

6.3.1 Experiment Settings

To investigate the role of fixed and float paths in model robustness, this section compares metrics of a VGG16 model trained on the CIFAR10 dataset with different defenses:

- Standard data without any defenses, denoted as STD.
- Noised samples with perturbation sizes of 0.125 and 0.25, denoted as Noise-125 and Noise-250.
- FGSM-perturbed samples with epsilon values of 4/255 and 8/255, denoted as FGSM-04 and FGSM-08.
- PGD-perturbed samples with epsilon values of 4/255 and 8/255, denoted as PGD-04 and PGD-08.

Each model is trained with the SGD optimizer for 120 epochs and a batch size of 256. A milestone learning rate scheduler is adopted, where the initial learning rate is 0.1 and is reduced by a factor of 0.1 after 60 and 90 epochs.

6.3.2 Scales and Directions

The first set of experiments compares the direction and scale of fixed and float paths in relation to adversarial examples. Given test data $(\mathbf{x}, y)$, the adversarial example is computed under an FGSM attack with epsilon 8/255 for each data point.

The values of the fixed paths $Z^I(\mathbf{x})$ and $Z^I(\mathbf{x}')$ are computed by deactivating the float neurons after identifying the fixed and float neurons between $\mathbf{x}$ and $\mathbf{x}'$. The differences between the fixed paths and float paths are calculated as:

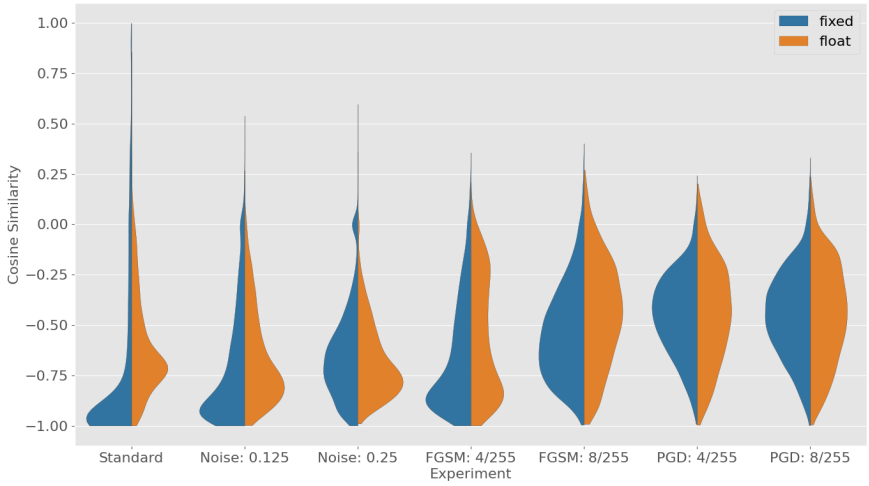
$$\begin{aligned} \delta^I(\mathbf{x}, \mathbf{x}') &= \mathcal{Z}^I(\mathbf{x}) - \mathcal{Z}^I(\mathbf{x}'), \\ \delta^T(\mathbf{x}, \mathbf{x}') &= [f(\mathbf{x}) - f(\mathbf{x}')] - \delta^I(\mathbf{x}, \mathbf{x}'), \end{aligned} \tag{6.9}$$

where $\mathcal{Z}^I(\mathbf{x})$ and $\mathcal{Z}^I(\mathbf{x}')$ are the sums of all fixed paths between $\mathbf{x}, \mathbf{x}'$ at inputs $\mathbf{x}$ and $\mathbf{x}'$.

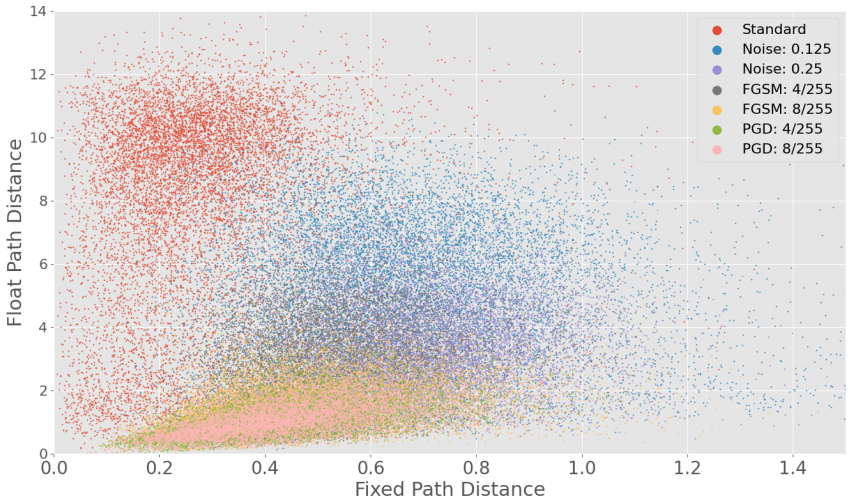
Figure 6.1 presents the direction and scale of the fixed difference $\delta^I(\mathbf{x}, \mathbf{x}')$ and float path $\delta^T(\mathbf{x}, \mathbf{x}')$. The objective of this experiment is to investigate how models respond to adversarial examples. Figure 6.1(a) shows the distribution of cosine similarity of fixed and float paths to the ground truth label:

$$\begin{aligned} \mathcal{CS}^I &= \text{Cosine Similarity}(\delta^I(\mathbf{x}, \mathbf{x}'), y) \\ \mathcal{CS}^T &= \text{Cosine Similarity}(\delta^T(\mathbf{x}, \mathbf{x}'), y) \end{aligned} \tag{6.10}$$

As cosine similarity can be viewed as a representation of the angle between vectors, lower similarity suggests that the vectors diverge significantly in direction. By decomposing $f(\mathbf{x}) - f(\mathbf{x}')$ and comparing the cosine similarity, the effects of fixed paths and float paths on altering the prediction from a directional perspective can be described. Generally, as the robustness of the model increases, both $\mathcal{CS}^I$ and $\mathcal{CS}^T$ increase, indicating that the directional effects from both fixed and float paths on $f(\mathbf{x}') - f(\mathbf{x})$ are mitigated by robust training.



(a) Cosine Similarity of Fixed and Float Path to Label



(b) Distance of Fixed Path and Float Path

Figure 6.1: Direction and Scale of the fixed paths and float paths.

Comparing the cosine similarity between the fixed path and y, it can be observed that the cosine similarity between $\delta^I(\mathbf{x}, \mathbf{x}')$ and y clusters around -1 for the under-defended models (STD, Noise-125, and FGSM: 4/255), meaning that the fixed path moves directly in the opposite direction of the ground truth label. As the robustness

of the models increases, the distribution of $\mathcal{CS}^I$ shifts upward with less skewness. A similar observation can be made for $\mathcal{CS}^T$. The under-defended model has lower cosine similarity between $\delta^T(\mathbf{x})$ and y, while that of robust models is more evenly distributed around -0.4.

Figure 6.1(b) presents the l_2 norm of δ^I and δ^T. It is noteworthy that there are significant differences between the scales of the float path and the fixed path. For all models, the norm of δ^I is distributed within (0, 1), while that of δ^T varies from 0 to 16 for under-defended models and is still greater than $|\delta^I|$ for robust models. In the following sections, it is shown that for models with minor defenses, the proportion of float neurons as well as the local Lipschitz constant decreases dramatically. However, the scale of float paths δ^T remains high for most models. This implies that the robust accuracy provided by defenses results from reducing the fixed paths' distance, while the float paths still severely affect model robustness.

As supporting evidence, Table 6.1 shows the accuracy of raw input as well as the sum of fixed paths under our decomposition. The fixed path is computed the same way as above for both clean data and adversarial data. For all models, the accuracy on clean data is higher than the accuracy on the fixed path ($Z^I(\mathbf{x})$). However, the accuracy of the fixed path in adversarial examples is much higher than the accuracy with both float and fixed paths. In particular, the robust accuracy of the STD model is 0.0%, while it increases to 85.08% by removing the float path. In other words, a significant boost in accuracy is achieved by removing the float paths from adversarial examples. This implies that the float paths are primarily responsible for the model's vulnerability. We therefore categorize adversarial examples into float neuron vulnerable and Lipschitz vulnerable as follows:

Definition 9. *Let $\mathcal{N}$ be a neural network defined as in Definition 3. Given $\mathbf{x} \sim \mathbb{D}_x$, denote $\mathbf{x}' \in B(\mathbf{x}, r)$ as an adversarial example of $\mathbf{x}$ such that:*

$$\arg\max_{m \in Y} f_m(\mathbf{x}) \neq \arg\max_{m \in Y} f_m(\mathbf{x}'). \tag{6.11}$$

Table 6.1: Accuracy on clean and adversarial data, baseline prediction and fixed path.

			Noise		FGSM		PGD	
ϵ		Standard	0.125	0.25	4/255	8/255	4/255	8/255
Clean	$f(\mathbf{x})$	93.05	86.25	73.59	88.40	82.77	80.94	80.47
	$\mathcal{Z}^I(\mathbf{x})$	86.95	72.19	59.84	81.56	76.88	75.08	73.55
Adversarial	$f(\mathbf{x}')$	00.00	12.54	19.53	36.25	47.15	49.26	48.59
	$\mathcal{Z}^I(\mathbf{x}')$	85.08	65.90	50.98	75.86	70.23	67.58	66.84
Accuracy Gap		85.08	53.36	31.45	39.61	23.09	18.32	18.24
Float Vulnerable		9962	9825	9686	9069	8091	8669	8798

The difference between $f(\mathbf{x})$ *and* $f(\mathbf{x}')$ *can be written as:*

$$f(\mathbf{x}) - f(\mathbf{x}') = \delta^I(\mathbf{x}, \mathbf{x}') + \delta^T(\mathbf{x}, \mathbf{x}'), \tag{6.12}$$

where $\delta^I(\mathbf{x}, \mathbf{x}') = \mathcal{Z}^I(\mathbf{x}, \mathcal{A}; \mathcal{R}) - \mathcal{Z}^I(\mathbf{x}', \mathcal{A}'; \mathcal{R})$ *and* $\delta^T(\mathbf{x}, \mathbf{x}') = \mathcal{Z}^T(\mathbf{x}', \mathcal{A}; \mathcal{R}) - \mathcal{Z}^T(\mathbf{x}', \mathcal{A}'; \mathcal{R})$ *are the differences between the fixed path and float path defined on* $\mathcal{R} = \mathbf{x}, \mathbf{x}'$*. Denote* $\mathbf{x}$ *as Lipschitz vulnerable if* $|\delta^I(\mathbf{x}, \mathbf{x}')| > |\delta^T(\mathbf{x}, \mathbf{x}')|$*; otherwise, it is float vulnerable.*

The last row in Table 6.1 shows the number of float vulnerable data points from the test dataset across different models. For the Standard model, most of the test instances are float vulnerable. On the other hand, training with noised samples can improve the robustness of the model but fails to address the fact that most of the test samples remain float vulnerable. In fact, Figures 6.3 and 6.4 suggest that the increased robustness of models trained with noised data results from the reduction of the Lipschitz constant. Lastly, a noteworthy decrease in float vulnerable data can be observed in models trained with adversarial examples (FGSM/PGD-04/08), although still over 80% of test samples respond negatively to the float path.

6.3.3 Two Types of Vulnerabilities

This section further investigates the two types of vulnerabilities by comparing the differences between models.

6.3.3.1 Float Vulnerability

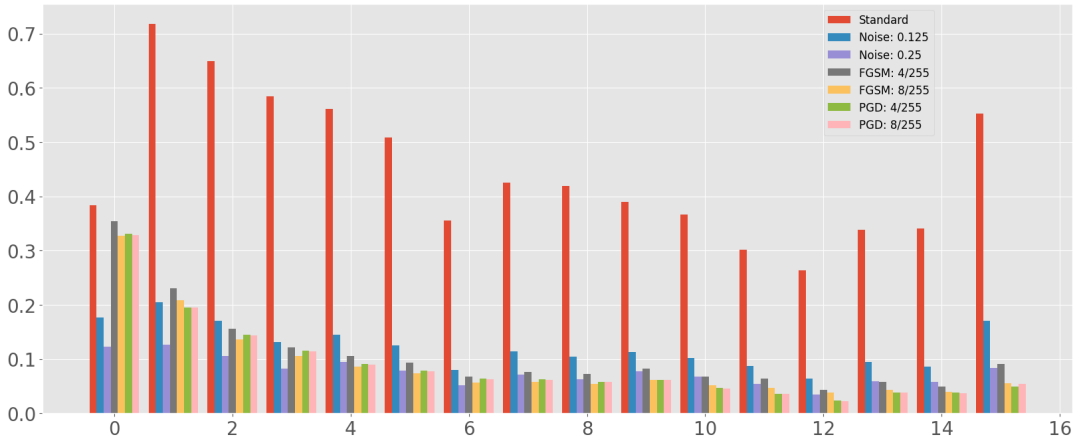


Figure 6.2: The average proportion of float neuron VGG16. All the networks are trained on the CIFAR10 dataset for 120 epochs. We test the first 1000 data from the validation dataset. For each of the test data, we generate a noised dataset with the size of 1000 samples and perturbation size of 4/255 under $l2$ bound.

Figure 6.2 presents the proportion of float neurons at different layers for the VGG16 network trained with various defenses. The models are trained as described in Section 6.3.1.

We estimate the proportion of float neurons for the first 1000 samples from the validation set. For each data point, 1000 noised data points with a perturbation size of 8/255 under the l_∞ norm are generated. This allows us to assess the local stability of different neural networks.

It is noteworthy that the network without any defenses has the highest number of float neurons. In particular, 70% of neurons in layer 1 are float neurons, indicating that the network with standard training is locally unstable. As the network goes

deeper, the number of float neurons decreases, which aligns with our previous discussion.

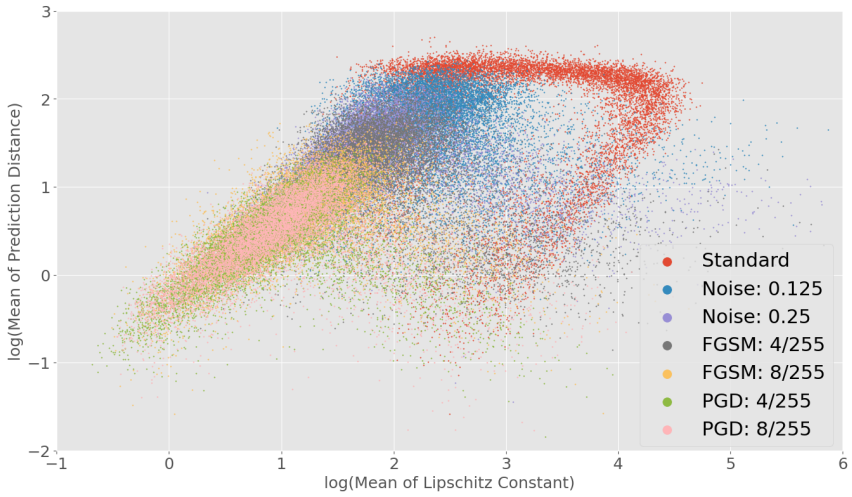
In comparison with standard training, networks trained with noised data have relatively fewer float neurons. The noised data are perturbed by Gaussian noise $N(0, \sigma I)$ with a variance of σ. The average ratio of float neurons across different layers is around 15% for $\sigma = 0.12$ and 10% for $\sigma = 0.25$. This indicates that even minor defenses can dramatically reduce the number of float neurons locally.

At the same time, adversarial training can also stabilize local neurons. Unlike noised training, FGSM reduces the number of float neurons in deep layers while increasing the number in shallow layers. In layers 0 to 4, the proportion of float neurons in FGSM/PGD-04 and FGSM/PGD-08 models is higher than in Noised ($\sigma = 0.12$) and Noised ($\sigma = 0.25$) models. However, in deeper layers, there are fewer float neurons in models trained with adversarial examples. Specifically, in layers deeper than 12, the ratio of float neurons in PGD models drops to less than 5

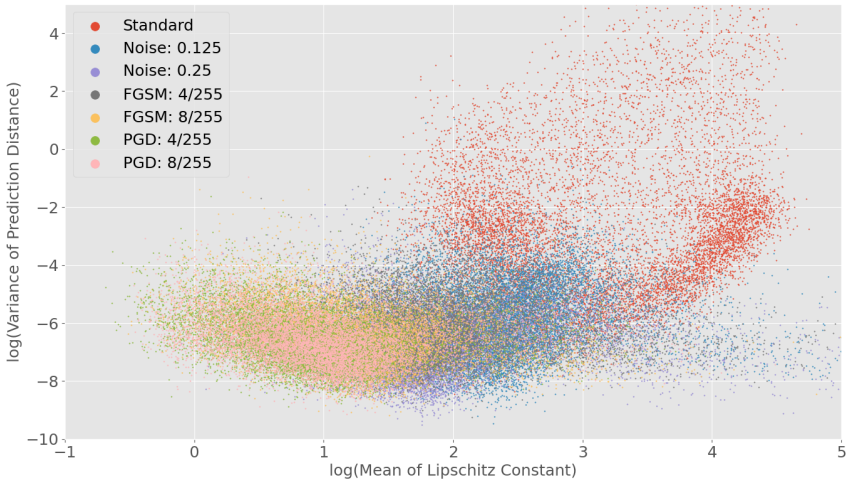
Overall, Figure 6.2 suggests that models with stronger defenses tend to have fewer float neurons, especially in deep layers. Moreover, the differences between the undefended model and defended models, even those with weaker defenses, are distinguishable.

Lipschitz Vulnerability This section investigates how the Lipschitz constant affects the prediction difference between $\mathbf{x}$ and $\mathbf{x} + \boldsymbol{\epsilon} \in B(\mathbf{x}, r)$. The models are trained as described in Section 6.3.1. For each model, 500 noise points $\mathbf{x} + \boldsymbol{\epsilon}$ are sampled around $\mathbf{x}$ with an l_∞ bound of $|\boldsymbol{\epsilon}|_\infty \leq 8/255$. For each of the noised samples, the Lipschitz constant at $\mathbf{x} + \boldsymbol{\epsilon}$ as well as the adversarial example of $\mathbf{x} + \boldsymbol{\epsilon}$ are computed under an FGSM-08 attack.

Figure 6.3 presents the relationship between the local Lipschitz constant and prediction difference. In each plot, the x-axis represents the log mean of the Lipschitz constant for 500 perturbed samples $\mathbf{x}' \in B(\mathbf{x}, r)$, while the y-axis shows (a) the log mean of prediction difference between $\mathbf{x}$ and $\mathbf{x}'$, (b) the log variance of prediction difference, and (c) the variance of the Lipschitz constant. The objective of this



(a) log(Mean of Prediction Distance)



(b) log(Variance of Prediction Distance)

Figure 6.3: The difference between $f(x)$ and $f(x')$ are lower and less volatile for model with higher robustness. In each figure, x-axis is the log mean of Lipschitz constant on 500 test samples, and y-axis are (a) The log mean of prediction distance, (b) variance of prediction distance.

section is to briefly illustrate the local stability of neural networks under different models.

Figure 6.3(a) shows the relationship between the averaged Lipschitz constant and the prediction difference of FGSM adversarial examples. For the Clean model,

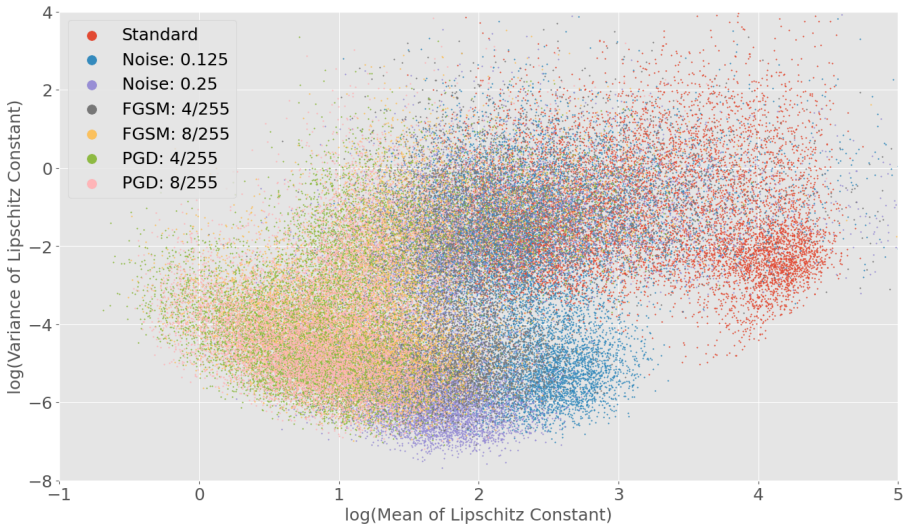


Figure 6.4: The mean and variance of local Lipschitz constant.

both the Lipschitz constant and prediction difference are higher than those of other models. Compared with properly defended models (FGSM-08, PGD-04/08), under-defended models (Noise-0.125/0.25, FGSM-04) exhibit higher prediction differences. This is due to the following factors: first, a higher Lipschitz constant leads to a faster change in the fixed path, implying higher Lipschitz vulnerability. Second, more diverged scatters suggest that the relationship between prediction difference and the Lipschitz constant is highly non-linear. Given that the distance between the input and adversarial example is fixed, this implies that float paths significantly alter the prediction result.

Figure 6.3(b) compares the variance of the Lipschitz constant with the change in prediction between $\mathbf{x} + \epsilon$ and its adversarial example. Figure 6.4 compares the mean and variance of the Lipschitz constant. Similar to prior results, models with defenses are more stable locally, as they tend to have lower variance in prediction change and lower variance in the Lipschitz constant. However, the difference between under-defended models and properly defended models is not significant. In fact, although Noise-0.125 and Noise-0.25 models have higher averaged Lipschitz constants, the variance of both the Lipschitz constant and prediction change is in the same magnitude as the properly defended models.

By investigating the local relationship between the Lipschitz constant and prediction change, it is found that robust training can greatly tighten the local Lipschitz properties and stabilize the local performance of models, regardless of the method used. Moreover, an additional boost in model robustness provided by adversarial training comes from reducing the effect of the float path.

6.4 Float Vulnerabilities and Smooth Classifier

6.4.1 Randomized Certifiable Classifier

The research on *certifiable training* aims to provide a guaranteed region for its input $\mathbf{x}$, within which a classifier always provides consistent results. Specifically, a classifier is regarded as robust for an input $\mathbf{x}$ against perturbations of size r if:

$$\arg\max_{m \in Y} f_m(x') = \arg\max_{m} f_m(x), \forall x' \in B_p(x, r) \tag{6.13}$$

where $B_p(x, r) := \{x' : \|x' - x\|_p \leq r\}$ is the sphere with radius r measured by the metric induced by p-norm. By taking the standard performance of the classifier into account, the robust accuracy of f with radius r is then defined as:

$$R(f) = \mathbb{E}_{(x,y)\sim D}\left[\arg\max_{m \in Y} f_m(x') = y, \forall x' \in B_p(x, r)\right]. \tag{6.14}$$

However, robust models often incur increased stability and impaired expressive ability. As a concession for this, randomized algorithms are proposed to verify the network with a sound theoretical bound at the cost of slight additional computation, rather than compromising model performance.

Let g be a randomized algorithm constructed based on classifier f. Given data $(x, y) \sim D$, g employs a certain degree of randomness during the induction of f. For instance, the smoothed classifier g computes the probability that $f(x + \epsilon)$ belongs to class i, given $\epsilon \in \mathcal{N}(0, \sigma^2 I)$ ([241]):

$$g_i(x) = \mathbb{P}(\arg\max_{m \in \mathcal{Y}} f_m(x + \epsilon) = i), \epsilon \sim \mathcal{N}(0, \sigma^2 I), \tag{6.15}$$

With a certain confidence level α, the lower bound $\underline{p_A}$ of the random variable $g_y(x)$ and the upper bound $\overline{p_B}$ of the probability of the second most probable class $\max_{m \in Y} g_m(x)$ can be computed. This induces a certified radius of classifier $g(x)$:

$$r = \frac{\sigma}{2}\big(\Phi^{-1}(\underline{p_A}) - \Phi^{-1}(\overline{p_B})\big). \tag{6.16}$$

For every $x' \in B_2(x, r)$, if $\arg\max_{m \in Y} g_m(x) = y$, then $\arg\max_{m \in Y} g_m(x') = y$. In other words, with a confidence level of α, every x' within the radius can be correctly classified by the smoothed classifier. The smoothed classifier can also appear in other forms depending on the randomness and training algorithm ([243], [245], [247]).

6.4.2 Float Path and Network Robustness

Consider $(x, y) \sim D$ has extreme large Lipschitz constant in $B(x, r)$, then there exists $x' \in B(x, r)$ such that $f(x') - f(x) \gg f_y(x) - f_{m \neq y}(x)$. We refer such an x as a Lipschitz vulnerable data. For example, the Lipschitz constant of a large scale network without any defenses up to hundreds, which means small perturbation ϵ can alter the prediction of $f(x+\epsilon)$. By applying the randomized smoothing on Theorem 8, it is shown that the smoothed classifier fails to correct such data.

Theorem 9. *Let f be the base classifier. Given $(x, y) \sim D$ with $f(x) = y$, denote $M(f(x), y) := min_{y' \neq y} |f(x)_y - f(x)'_y|$ as the margin operator of prediction vector. If*

$$\|J(x)\| > \frac{M(f(x), y) + \mathbb{E}[M(Z^T(x, \mathcal{A}; \mathcal{R}), y)]}{r},$$

then for any smoothed classifier g defined as above, there exist $x' \in B(x, r)$ such that $g(x') \neq y$.

Proof (Theorem 9)**.** *Statement 2 of Theorem 8 suggests:*

$$f(x) - f(x') = J(x)(x - x') + \mathcal{Z}^T(x', \mathcal{A}; \mathcal{R}) - \mathcal{Z}^T(x', \mathcal{A}'; \mathcal{R}) \tag{6.17}$$

As $\|J(x)\| > \frac{M(f(x),y) + \mathcal{M}(\mathbb{E}[Z^T(x,\mathcal{A};\mathcal{R})], y)}{r}$, there exist x' and i such that:

$$f_i(x') > f_y(x') + \mathbb{E}[Z_y^T(x, \mathcal{A}; \mathcal{R}) - Z_i^T(x, \mathcal{A}; \mathcal{R})] \tag{6.18}$$

Therefore,

$$\begin{aligned} g_i(x) - g_i(x') > g_y(x) - g_y(x') + M(f(x), y), \\ g_i(x') > g_y \end{aligned} \tag{6.19}$$

On the other hand, consider $x' \in B(x, r)$ as an adversarial example that is misclassified by f but corrected by g. Then, given $\epsilon \sim D_{noise}$, $g_y(x' + \epsilon) > \arg\max_{m \in Y} g_m(x' + \epsilon)$. This means that the adversarial example x' appears by chance, while the majority of its neighbors are still voting for the correct label. In other words, the instability of the float path causes the misclassification. Such data is referred to as *float neuron vulnerable*.

The above discussion suggests that the smoothed classifier fails to boost the performance of Lipschitz-vulnerable examples but is able to correct float neuron vulnerable data by smoothing the sudden change of the float path in a region. Based on this, the SCRFP-2 is introduced in Algorithm 1.

Figure 6.5 illustrates the SC-RFP by showing the prediction, fixed path, and float path. We first present the prediction and the fixed path value in Figure 6.5(a). The float path value is then computed as $Z^T(x, \mathcal{A}; \mathcal{R}) = f(x) - Z^I(x, \mathcal{A}; \mathcal{R})$ in Figure 6.5(b). Finally, by repressing the float value, a locally stable prediction can be obtained from SCRFP-2.

6.4.3 Reforming the Float Paths

The discussion in above section suggests that the essence of smooth classifier is to average the float paths $\mathcal{Z}^T$ such that removes the float vulnerabilities of unsecured data. Based on this idea, this section introduces a downstream application name SCRFP-2 (Smoothed classifier with Reformed Float Path in Dual Direction) that is built on the smoothed classifier. As the suggested by the name, SCRFP-2 tunes the float path during both backward and forward pass to improve the robustness of neural network. The details and the objectives of the proposed method are discussed as follows.

6.4.3.1 Training with Amplified Float Paths

Algorithm 2: Train with SCRFP-2

Input: Network $\mathcal{N}$ with parameter $\boldsymbol{\theta}$, train set D_{train}, noise distribution $\mathcal{D}_{\text{noise}}$, amplification factor η_1

Function `Train`**:**

- **for** *Epoch* $t = 1 \dots T$ **do**
  - **while** $(\mathbf{x}, y) \in \mathcal{D}_{train}$ **do**
    - $\epsilon \leftarrow \mathcal{D}_{noise}$;
    - $\mathbf{x}_c, \mathbf{x}_n \leftarrow \mathbf{x}, \mathbf{x} + \epsilon$;
    - **for** *Block* i *in* $1, 2, \dots, d-1$ **do**
      - $\mathbf{z}_c, \mathbf{z}_n \leftarrow \psi_i \circ \phi_i(\mathbf{x}_c), \psi_i \circ \phi_i(\mathbf{x}_n)$;
      - $\mathbf{x}_c, \mathbf{x}_n \leftarrow \sigma(\mathbf{z}_c), \sigma(\mathbf{z}(x_n))$;
      - $\mathcal{I}_{ij}^T \leftarrow \hat{a}(\mathbf{z}_c) \neq \hat{a}(\mathbf{z}_n)$;
      - $\mathbf{z} \leftarrow \psi_i \circ \phi_i(\mathbf{x})$;
      - $\mathbf{x} \leftarrow \mathbf{z} + \mathbf{z} \cdot \mathcal{I}_i^T \times \eta_1$;
    - **end**
    - $\hat{y} \leftarrow \psi_d \circ \phi_d(\mathbf{x})$;
    - $\theta \leftarrow \boldsymbol{\theta} - \nabla_{\boldsymbol{\theta}} loss(\hat{y}, y)$;
  - **end**
- **end**

EndFunction

The illustration in Chapter 6.3 suggests that any robust training can significantly reduce the Lipschitz constant of a network. The lower Lipschitz constant compresses the float paths but still fails to correct the float vulnerability. Moreover, comparing models trained with noisy data to those trained with adversarial examples, it is found that FGSM/PGD models have less volatile neurons in deeper layers, which results in fewer float-vulnerable examples in the test dataset. Therefore, the objective during the training stage is to reduce the effects of float neurons on prediction.

The Train function in Algorithm 2 describes how the float paths are reformed during training. Compared with the standard forward process, the pre-activation value on the float path in the proposed method has a larger norm. In other words, SCRFP-2 rebalances the computational graph during the forward pass so that the float path contributes more to its prediction as well as the loss function. Therefore, during backward propagation, extra penalties are applied to the float path to reduce the effect of float neurons.

The intuition behind the above algorithm comes from the results in Table 6.1, which show that removing the float path has limited effects on the accuracy of clean data x, while it can greatly boost the accuracy of vulnerable examples around x. In other words, the float paths cause local instability in the prediction result. By amplifying the value of the float path, additional penalties are added to the float path to reduce its scale, thereby increasing the robustness of the model.

In Section 6.5, it is shown that amplifying the float path and repressing the fixed path can achieve similar results in boosting model robustness. However, Figure 6.2 suggests that there are fewer float neurons under robust training methods. Empirically, SCRFP-2 manipulates the float path value to avoid the explosion or vanishing of the prediction.

6.4.3.2 Prediction with Repressed Float Paths

On the other hand, the prediction of SCRFP-2 applies a randomized smoothing algorithm but represses the value of float paths. For vulnerable data, adding back the float paths causes a decline in accuracy, which suggests that the float paths negatively contribute to the prediction. Therefore, a repression factor η_2 is introduced to reduce the effect of float paths. Since the float part of a network is unstable, randomized smoothing can be applied to average the result.

A randomized classifier g is constructed based on classifier f. Given data $(\mathbf{x}, y) \sim \mathcal{D}$, g employs a certain degree of randomness during the induction of f. For instance, the smoothed classifier g computes the probability that $f(\mathbf{x} + \boldsymbol{\epsilon})$ belongs to class i

Algorithm 3: Predict with SCRFP-2

Input: Network $\mathcal{N}$ with parameter $\boldsymbol{\theta}$, noise distribution $\mathcal{D}_{\text{noise}}$, randomized algorithm g, repress factor η_2, test input $\mathbf{x}$

Function `Predict(x)`**:**
  Initialize prediction counter pc with size c;
  while g *sample noise* ϵ **do**
    for *Block* i *in* $1, 2, \ldots, d-1$ **do**
      $\mathbf{z}_s, \mathbf{z}_n \leftarrow \psi_i \circ \phi_i(\mathbf{x}_c), \psi_i \circ \phi_i(\mathbf{x}_n)$;
      $\mathcal{I}_{ij}^T \leftarrow \hat{a}(\mathbf{z}_c) \neq \hat{a}(\mathbf{z}_n)$;
      $\mathbf{z}_n \leftarrow \mathbf{z}_n - \mathbf{z}_n \cdot \mathcal{I}_i^T \times \eta_2$;
      $\mathbf{x}_c, \mathbf{x}_n \leftarrow \sigma(\mathbf{z}_c), \sigma(\mathbf{z}_n)$;
    end
    $pred_n \leftarrow \arg\max(\psi_d \circ \phi_d(\mathbf{x}_n))$;
    $pc[pred_n] \leftarrow pc[pred_n] + 1$
  end
  $\hat{c}_A, \hat{c}_B \leftarrow \arg\max(pc, \text{top}_k = 2)$;
  $\hat{n}_A, \hat{n}_B \leftarrow pc[\hat{c}_A], pc[\hat{c}_B]$;
  if `BinomPValue`$(n_A, n_A + n_B, 0.5) \leq \alpha$ **then return** $\hat{c}_A$;
  else return *Abstain*;
EndFunction

given $\boldsymbol{\epsilon} \in \mathcal{N}(0, \sigma^2 I)$ [241]:

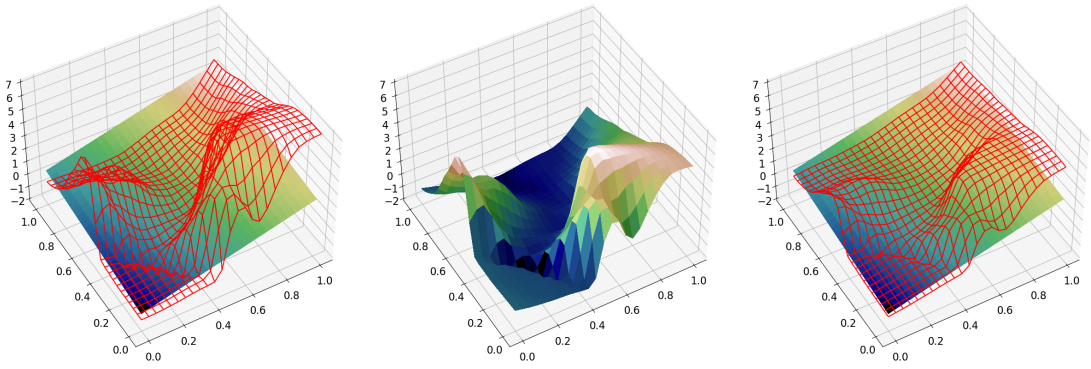
$$g_i(\mathbf{x}) = \mathbb{P}(\arg\max_{m \in \mathcal{Y}} f_m(\mathbf{x} + \boldsymbol{\epsilon}) = i), \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 I), \tag{6.20}$$

With certain confidence level α, the lower bound of $\underline{p_A}$ of random variable $g_y(\mathbf{x})$ and the upper bound $\overline{p_B}$ of the probability of second possible class $\max m \in Y g_y(\mathbf{x})$ can be computed. This induces a certified radius of classifier $g(\mathbf{x})$:

$$r = \frac{\sigma}{2}\big(\Phi^{-1}\big(\underline{p_A}\big) - \Phi^{-1}\big(\overline{p_B}\big)\big). \tag{6.21}$$

For every $\mathbf{x}' \in B_2(\mathbf{x}, r)$, if $\arg\max_{m \in Y} g_m(\mathbf{x}) = y$, then $\arg\max_{m \in Y} g_m(\mathbf{x}') = y$. In

other words, with a confidence level of α, every $\mathbf{x}'$ within the radius can be correctly classified by the smoothed classifier. The smoothed classifier can also appear in other forms depending on the randomness and training algorithm [243], [245], [247].

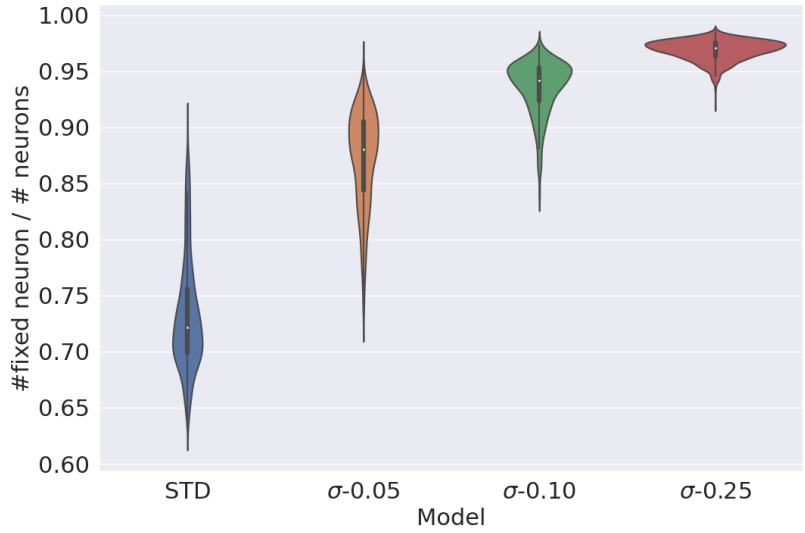


(a) Prediction Surface (b) Float Computation (c) SC-RFP Surface

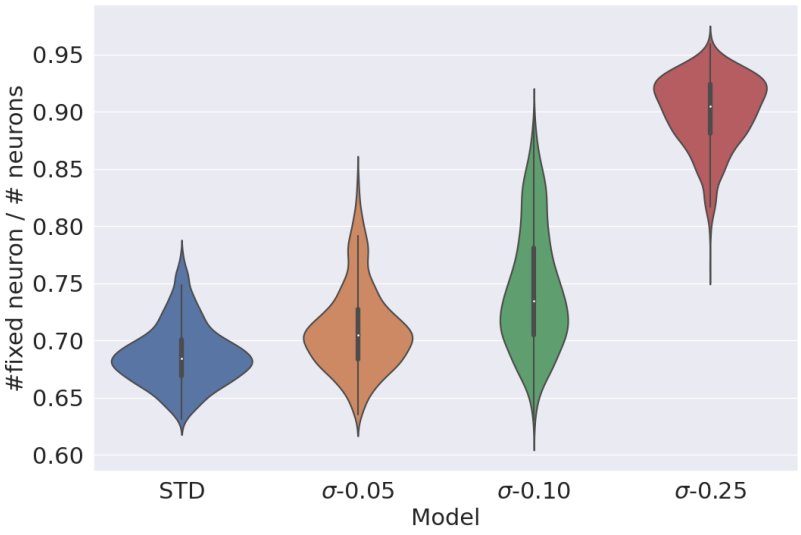
Figure 6.5: The value of first element of prediction vector from VGG16 model trained on CIFAR10 given a 2D slice centered at a random data from test set. (a) The wireframe represents the prediction $f(x)$ while the surface is the sum of fixed path $Z^I(x, \mathcal{A}; \mathcal{R})$, respectively. (b) The sum of float path $Z^T(x, \mathcal{A}; \mathcal{R}) = f(x) - Z^I(x, \mathcal{A}; \mathcal{R})$. (c) The wireframe and surface are prediction of SC-RFP: $Z^I(x, \mathcal{A}; \mathcal{R}) + \eta Z^T(x, \mathcal{A}; \mathcal{R})$ with $\eta < 1$ and sum of fixed path same as (a).

Figure 6.5 illustrates the SC-RFP by showing the prediction, fixed path, and float path. We first present the prediction and the fixed path value in Figure 6.5(a). The float path value is then computed as $Z^T(x, \mathcal{A}; \mathcal{R}) = f(x) - Z^I(x, \mathcal{A}; \mathcal{R})$ in Figure 6.5(b). Finally, by repressing the float value, we achieve a locally stable prediction from SC-RFP.

Intuitively, manipulating the computational graph can greatly change the prediction result, but we show that there is only a small proportion of paths affected by our method. Figure 6.6 presents the proportion of fixed neurons between x and $x + \epsilon$ in a VGG16 network trained on CIFAR10. It shows that models trained with noisy samples have a relatively more stable activation pattern, as the ratio of fixed neurons is lower. In particular, the model trained with clean data has an average



(a) $\epsilon \sim N(0, 0.1)$



(b) $\epsilon \sim N(0, 0.25)$

Figure 6.6: Proportion of fixed neuron between x and $x + \epsilon$ for VGG16 models trained on CIFAR10 with noised data at different scales: clean data, $\sigma = 0.05$, $\sigma = 0.10$ and $\sigma = 0.25$. Figure 6.6(a) and 6.6(b) show the ratio given $\epsilon \sim N(0, 0.1)$ and $\epsilon \sim N(0, 0.25)$.

fixed ratio of around 72%, while after adding a minimum scale of noise $\sigma = 0.05$, this increases to around 90%.

6.4.4 Certifiable Bound

The Predict function in Algorithm 3 shows the prediction step of SCRFP-2 under the scheme of randomized smoothing. The following theorem provides a certifiable boundary for the proposed algorithm, as previous works have done.

Theorem 10. *Let $\mathcal{N}$ be a network defined as Chapter 4.1. Let g be a smoothed classifier that samples noise from distribution D_{noise} and g' is SCRFP-2 built on g. Assume that the direction of $\boldsymbol{\epsilon}$ is uniformly distributed:*

$$\forall \|\boldsymbol{\xi}_1\| = \|\boldsymbol{\xi}_2\| = 1, P(\frac{\boldsymbol{\epsilon}}{\|\boldsymbol{\epsilon}\|} = \boldsymbol{\xi}_1) = P(\frac{\boldsymbol{\epsilon}}{\|\boldsymbol{\epsilon}\|} = \boldsymbol{\xi}_2). \tag{6.22}$$

If $\arg\max_{m \in Y} f_y(\mathbf{x}) = y$, *then*

$$\underline{p'_A} > \underline{p_A}, \quad \overline{p_B}' < \overline{p_B}, \tag{6.23}$$

where $\underline{p'_A}, \underline{p_A}$ are the lower bound of $g'_y(\mathbf{x})$ and $g_y(\mathbf{x})$, $\overline{p_B}', \overline{p_B}$ are the upper bound of $g'_{m \neq y}(\mathbf{x})$, $g'_{m \neq y}(\mathbf{x})$. Moreover, $\arg\max_{m \in Y} g'(\mathbf{x}) = y$ *for all* $\|\boldsymbol{\epsilon}\| \leq R$,

$$R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p_A}) - \Phi^{-1}(\overline{p_B})) \tag{6.24}$$

Notice that in the above theorem, the randomized classifier g is not specified. Instead, a constraint on the randomness of g is introduced to impose a constraint on the way SCRFP-2 averages the perturbation from float paths. This means that the theorem holds not only for the naive smoothed classifier but also for its variants. However, it does not hold for classifiers that sample directed noise.

The proof of Theorem 10 can be divided into three steps. First, given $\epsilon \sim D_{noise}$, the expectation of fixed paths for any $\mathbb{E}(Z^X(x+\epsilon))$ equals $Z^X(x)$. This means that the fixed part of g as well as g' equals that of $f(x)$. Next, we discuss how the float paths affect the prediction $f(x+\epsilon)$ and prove that repressing the float path can increase the expectation of $g'_y(x)$ and its variance. Using Chebyshev's Inequality, it

can be concluded that the lower bound p'_A is larger than that of p_A, and similarly for p'_B. Since both g and g' are random functions that sample noise from the same distribution, the certifiable radius holds for both g and g'.

Lemma 13. *Denote N as a neural network as Definition 4 with mapping function f, ζ is a fixed path in $B_2(x,p)$. Assume g is a randomized algorithm that applies noise ϵ from certain distribution D_{noise} on the computation path of ζ:*

$$g(\zeta(x)) = \zeta_{d,i}(\zeta_{i,0}(x) + \epsilon), i \in \{0, 1, \ldots, d-1\}, \epsilon \sim D_{noise} \tag{6.25}$$

If the direction of ϵ is uniformly distributed, then the expectation of $g(\zeta(x))$ is equals to $\zeta(x)$, that is:

$$\forall \|\eta_1\| = \|\eta_2\| = 1, P(\frac{\epsilon}{\|\epsilon\|} = \eta_1) = P(\frac{\epsilon}{\|\epsilon\|} = \eta_2) \Rightarrow E[g(\zeta(x))] = \zeta(x), \tag{6.26}$$

where ϵ is the noise generated for randomized algorithm g.

Proof. *Assume that g samples noise at layer k. Then for any g, $\zeta_(k,0) = g(\zeta_{k,0})$, which is denoted as x_k. Since the activation function is piece-wise linear, the mapping of $\zeta_{d,k}(\cdot)$ is linear. We have:*

$$\sum_{i}^{n_k} \frac{\partial^2 y_j}{\partial z_{kj}^2} = 0. \tag{6.27}$$

$\zeta_{d,k}$ is harmonic. Therefore, given any $B(x,r)$ with radius $r > 0$:

$$\zeta_{d,k}(x) = \frac{1}{n\omega_n r^{n-1}} \int_{\partial B(x,r)} \zeta d\sigma, \tag{6.28}$$

where ω_n is the volume of the unit ball in n dimensions, σ is the $(n-1)$ dimensional surface measure. Consider the expectation of $g(x)$:

$$\begin{aligned} \mathbb{E}(g(x)) &= \int p(\epsilon)\zeta_{d,k}(x_k + \epsilon)d\mu \\ &= \int \int_{\|\epsilon\|=r} p(\epsilon)\zeta_{d,k}(x_k + \epsilon)d\mu dr, \end{aligned} \tag{6.29}$$

where μ is the probability measure of ϵ D_{noise}. As the noise sampling is assumed to be direction irrelevant, the measure of $p(\epsilon | \|\epsilon\|_2 = r)$ is uniformly distributed given

radius r. Therefore:

$$\int_{\|\epsilon\|=r} p(\epsilon)\zeta_{d,k}d\mu = P(\|\epsilon\|_2 = r)\zeta_{d,k}. \tag{6.30}$$

Equation 6.29 then equals:

$$\begin{aligned}\mathbb{E}(g(x)) &= \int p(\epsilon)\zeta_{d,k}(x_k+\epsilon)d\mu \\ &= \int P(\|\epsilon\|_2 = r)\zeta_{d,k}dr \\ &= \zeta_{d,k}(x)\end{aligned} \tag{6.31}$$

Lemma 14. *Let f and g be the mapping function and randomized classifier defined as above. Given radius r such that, almost surely, $\mathbb{Z}^T(B(x,r)) = \emptyset, \forall(x,y) \sim D$, then the accuracy of base classifier and the naive smoothed classifier are same, that is:*

$$\mathbb{E}_{(x,y)\sim D}\left[\arg\max_{m\in Y} g_m(x) = y\right] = \mathbb{E}_{(x,y)\sim D}\left[\arg\max_{m\in Y} f_m(x) = y\right]$$

Lemma **??** is achieved by directly applying Lemma 13 to all the computational graphs of the network. It shows that if all the neurons have locally stable activation patterns with respect to the distribution of the dataset, the smoothed classifier provides identical accuracy to the base classifier. With Lemma **??**, the proof of Theorem 9 is presented as follows.

Proof (Theorem reftheorem.false)**.** *Statement 4 of 8 suggests:*

$$f(x) - f(x') = J(x)(x-x') + \mathcal{Z}^T(x', \mathcal{A}; \mathcal{R}) - \mathcal{Z}^T(x', \mathcal{A}'; \mathcal{R}) \tag{6.32}$$

As $\|J(x)\| > \frac{M(f(x),y)+\mathcal{M}(\mathbb{E}[Z^T(x,\mathcal{A};\mathcal{R})],y)}{r}$*, there exist x' and i such that:*

$$f_i(x') > f_y(x') + \mathbb{E}[Z_y^T(x, \mathcal{A}; \mathcal{R}) - Z_i^T(x, \mathcal{A}; \mathcal{R})] \tag{6.33}$$

Therefore,

$$\begin{aligned} g_i(x) - g_i(x') &> g_y(x) - g_y(x') + M(f(x), y), \\ g_i(x') &> g_y \end{aligned} \tag{6.34}$$

The next step of the discussion is to consider the case where x is an adversarial example that misleads f but is correctly classified by g, referred to as a float neuron vulnerable example. The following theorem suggests that the float path at x is the cause of the altered prediction. In other words, the network is locally correct around x, but there is a sudden change in the float path that causes the misclassification.

Theorem 11. *Let f and g be the base and smoothed classifiers defined above. Given $(x, y) \sim D$, denote $x' \in B(x, r)$ as an adversarial example that misleads the base classifier but is corrected by g without abstaining:*

$$\arg\max_{m \in Y} f_m(x) = i, \quad \arg\max_{m \in Y} g_m(x) = y, i \neq y \tag{6.35}$$

then loss of float path is higher than that of g:

$$CE(\mathcal{Z}^T(x', \mathcal{A}; \mathcal{R}), onehot(y)) > CE(g(\mathcal{Z}^T(x', \mathcal{A}; \mathcal{R})), onehot(y)) \tag{6.36}$$

where $CE(\cdot, \cdot)$ is the cross entropy loss, onehot(y) is the one hot embedding of label y.

Proof (Theorem 11)**.** *Since* $\arg\max_{m \in Y} f_m(x) \neq y$, $\arg\max_{m \in Y} g_m(x) = y$, *there exist* $i \in \{1, \ldots, c\}$*:*

$$\begin{aligned} g_y(x) &> f_y(x) \\ g_i(x) &< f_i(x) \end{aligned} \tag{6.37}$$

Then it can be show that:

$$g_y(x) - g_i(x) \geq f_y(x) - f_i(x) \tag{6.38}$$

Introducing Statement 4 of Lemma 8 and Lemma 14:

$$g_i(Z^T(x, \mathcal{A}; \mathcal{R})) - g_y(Z^T(x, \mathcal{A}; \mathcal{R})) < Z_i^T(x, \mathcal{A}; \mathcal{R}) - Z_y^T(x, \mathcal{A}; \mathcal{R}). \tag{6.39}$$

Moreover, since the prediction is not abstained.

$$\mathbb{E}[g_y(x)] - f_y(x) > \mathbb{E}[g_j(x)] > f_j(x), \forall j \neq i, y \tag{6.40}$$

This directly leads us to the result.

Theorem 9 and Theorem 11 show that a smoothed classifier is not able to boost the performance of the fixed path, but it is effective in reducing the sudden changes in the float path within a region. Specifically, if x' can be corrected by g, it results from the sudden change provided by the float path. A higher confidence score of $g(x')$ can be achieved by reducing the weight of the float path during the computation. On the other hand, if x' is Lipschitz vulnerable, then regardless of the form of the smoothed classifier, it cannot be corrected. This leads us to the theoretical basis of the SC-RFP algorithm.

6.4.5 Verifiable Radius

At the end of Section 6.4.2, Theorem 10 is introduced to describe the upper and lower bounds of SC-RFP as well as the certified radius. We present the proof of Theorem 10 below.

Theorem 12. *Let $\mathcal{N}$ be a network defined as Section 4.1. Let g be a smoothed classifier that samples noise from distribution D_{noise} and g' is the SC-RFP built on g. Assume that the direction of ϵ is uniformly distributed.*

$$\forall \|\eta_1\| = \|\eta_2\| = 1, P(\frac{\epsilon}{\|\epsilon\|} = \eta_1) = P(\frac{\epsilon}{\|\epsilon\|} = \eta_2), \epsilon \sim D \tag{6.41}$$

If $\arg\max_{m \in Y} f_y(x) = y$, *then*

$$\underline{p'_A} > \underline{p_A}, \overline{p'_B} < \overline{p_B}, \tag{6.42}$$

where $\underline{p'_A}, \underline{p_A}$ are the lower bound of $g'_y(x)$ and $g_y(x)$, $\overline{p'_B}, \overline{p_B}$ are the upper bound of $g'_{m \neq y}(x)$, $g'_{m \neq y}(x)$. Moreover, $\arg\max_{m \in Y} g'(x) = y$ *for all* $\|\epsilon\| \leq R$,

$$R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p_A}) - \Phi^{-1}(\overline{p_B})) \tag{6.43}$$

Proof. *The proof of Theorem 10 can be divided into three steps. First, we show that given $\epsilon \sim D_{noise}$, the expectation of fixed paths for any $\mathbb{E}(Z^X(x+\epsilon)) = Z^X(x)$. Consider ζ as a fixed path in a region R; then, from Lemma 13, we have:*

$$E[g(\zeta(x))] = \zeta(x). \tag{6.44}$$

Notice that, since the SC-RFP does not affect the sample of noise, then above equation holds for both g and g':

$$E[g'(\zeta(x))] = \zeta(x). \tag{6.45}$$

By applying randomized smoothing on statement 1 of Lemma 8, we have:

$$\begin{aligned} E(g(x)) &= \mathbb{E}(g(\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}))) + \mathbb{E}(g(\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}))), \\ E(g'(x)) &= \mathbb{E}(g'(\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}))) + \mathbb{E}(g'(\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}))), \end{aligned} \tag{6.46}$$

where we use $g'(\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}))$ to denote applying smoothing algorithm g on deterministic function $\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R})$. Since $\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R})$ is the aggregation of all the fixed path, with Equation 6.44 and 6.45, it can be shown that:

$$\mathbb{E}(g'(\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}))) = \mathbb{E}(g(\mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}))) = \mathcal{Z}^I(x, \mathcal{A}; \mathcal{R}) \tag{6.47}$$

This means that the difference between $\mathbb{E}(g(x))$ and $\mathbb{E}(g'(x))$ is the same as that between $g(\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}))$ and $g'(\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}))$.

The next step is to investigate how the float path affects the prediction of $g(\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}))$ and $g'(\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R}))$. Given $\epsilon \sim D_{noise}$, denote P_1 as the event that $f_y(x + \epsilon) > f_y(x)$. Since the fixed paths have been excluded, P_1 holding means that the float path boosts the probability of $f_y(x + \epsilon)$. However, as $\arg\max_{m \in Y} f_y(x) = y$, repressing the float path does not alter the prediction of $f_y(x')$. On the other hand, denote P_2 as the event that $f_y(x + \epsilon) < f_y(x)$. This means that the float path $\mathcal{Z}^T(x + \epsilon, \mathcal{A}; \mathcal{R}) < 0$ and negatively contributes to the prediction. Repressing the float path, therefore, can increase the probability of predicting y.

In other words, for any $x + \epsilon$, repressing the float path between x and x' can increase the predicted $f_y(x+\epsilon)$. We assume that P_1 and P_2 happen with probabilities p_1 and p_2, respectively. Then, given a certain number of samplings,

$$E(g'_y(x)) > E(g'_y(x)), Var(g'_y(x)) < Var(g_y(x))$$

By applying Chebyshev Inequality, the following results can be obtained:

$$\begin{aligned} \underline{p'_A} &> \underline{p_A} \\ \overline{p'_B} &< \overline{p_B} \end{aligned} \tag{6.48}$$

Therefore, the lower bound p'_A is larger than that of p_A, and similarly for p'_B.

Computing the certified radius of x is then the same as in [241]. Since both g and g' are random functions that sample noise from the same distribution, the Neyman-Pearson theorem holds for both g and g'. Therefore, the certified radius remains unchanged.

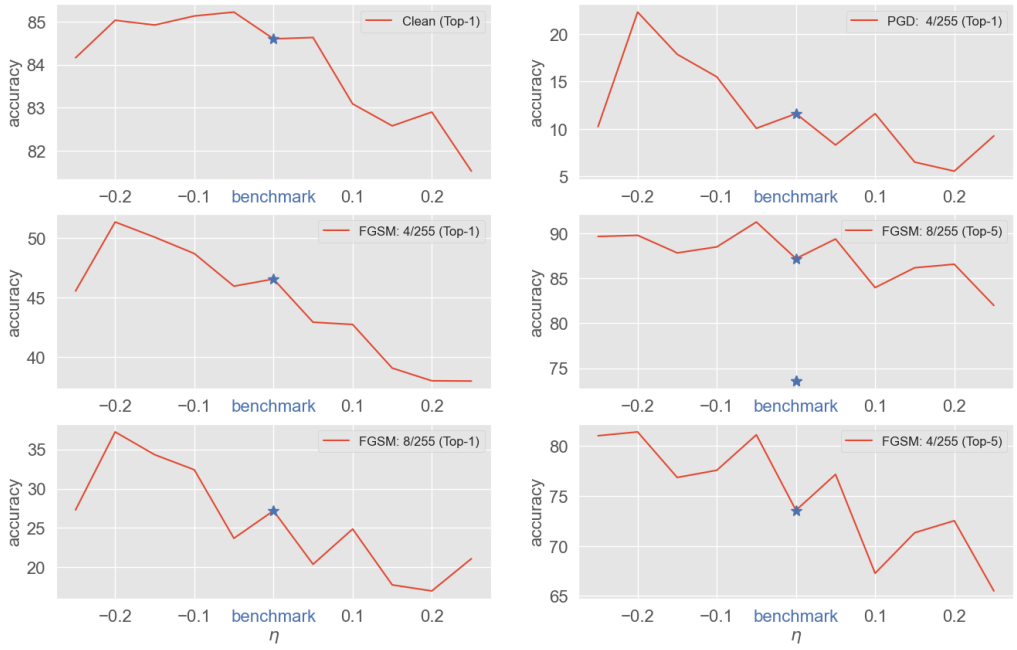
6.5 Experiment

In Section 6.2, the computational graph of a neural network is decomposed into float paths and fixed paths, where the float paths represent the instability of network predictions. The Smoothed Classifier with Reformed Float Path in Dual Direction (SCRFP-2) aims to improve the robustness of deep networks by manipulating the float path during both forward and backward propagation. The experiments in this section investigate the performance of SCRFP-2.

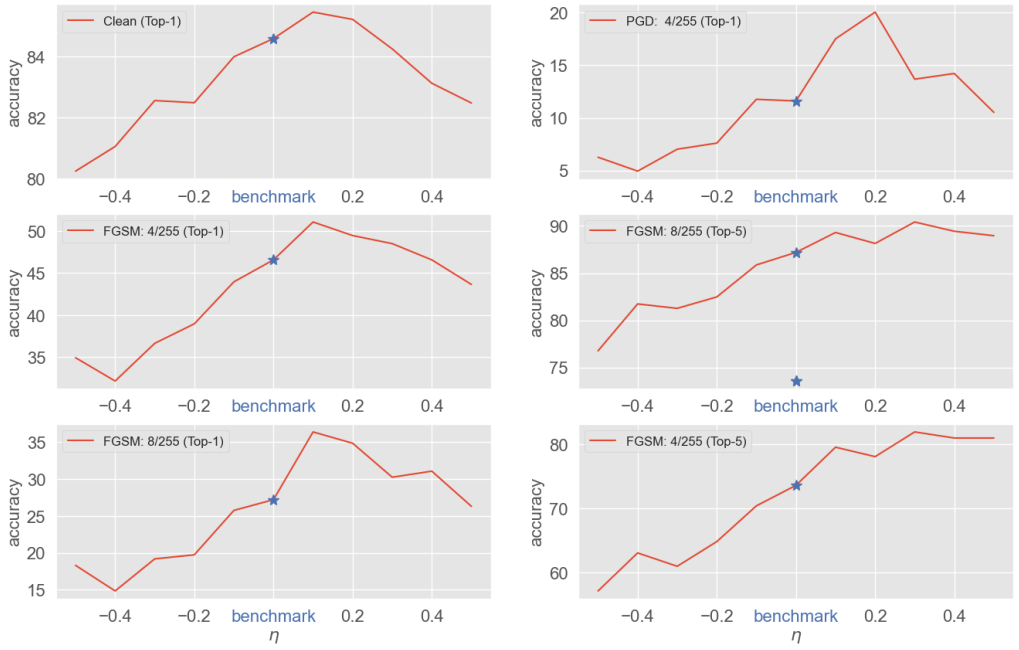
6.5.1 Training Stage

This section aims to illustrate the effect of the training stage of SCRFP-2 as presented in Algorithm 2. Given input $\mathbf{x}$, a noised counterpart $\mathbf{x} + \boldsymbol{\epsilon}$ is generated, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma I)$ is Gaussian noise with variance σ. At each block, the fixed and float neurons of the set $\mathbf{x}, \mathbf{x} + \boldsymbol{\epsilon}$ are computed. According to the status of the neurons, the pre-activation value is modified with a reform factor $\eta \in (-1, 1)$. For example, refactoring the value of fixed neurons with $\eta < 0$ will repress $\mathcal{Z}^I(\mathbf{x})$, while refactoring the value of float neurons with $\eta > 0$ will amplify $\mathcal{Z}^T(\mathbf{x})$.

Figures 6.7(a) and 6.7(b) compare the performance and robustness of models trained with refactored float/fixed paths given different values of η. Each model is trained on the CIFAR10 dataset for 120 epochs with a batch size of 128 using the SGD optimizer and a weight decay of $1e-4$. A milestone learning rate scheduler with an initial learning rate of 0.1 is used during training, where the learning rate is multiplied by 0.1 after 60 and 90 epochs. The model performance is reported by the



(a) Models Trained with Refactored Fixed Path



(b) Models Trained with Refactored Float Path

Figure 6.7: Performance and Robustness of VGG16 networks trained with refactored float path under different η_1.

clean accuracy of the data (upper left), while the top-1 and top-5 accuracy under FGSM and PGD attacks are chosen as proxies for model robustness (others).

Section 6.3 shows that fixed neurons make up the majority in models with defenses. To keep the two methods consistent at the level of modifying the network propagation, the range of η for the fixed path is lower than that for the float path. Each of the models is trained with the SGD optimizer for 120 epochs on the CIFAR10 dataset. A milestone learning rate scheduler is applied with an initial learning rate of 0.1, which is multiplied by 0.1 after 60 and 90 epochs.

It is notable that refactoring the computational path boosts the performance and robustness of the neural network compared to the benchmark. In fact, either repressing the value on the fixed path or amplifying the value on the float path can reduce the ratio of fixed values. During backward propagation, the network will then focus on minimizing the loss resulting from the unstable part of the network, which increases the robustness of the network and alleviates overfitting.

Given $\eta \in (0, 0.1)$, both the clean accuracy and robustness of the network increase, as shown in Figure 6.7(b). However, when $\eta > 0.2$, the computational graph of the network is heavily affected, which results in impaired performance. On the other hand, as the ratio of fixed neurons is higher than that of float neurons, the network is more sensitive to the manipulation of fixed neurons. Therefore, in practice, SCRFP-2 reforms values on the paths other than fixed paths during both forward and backward propagation.

6.5.2 CIFAR10

This section applies SCRFP-2 to the CIFAR10 dataset and compares it with randomized smoothing to show that both the training and validation steps of the proposed method can boost the performance of the benchmark model. Figures 6.8 to 6.10 show the certified accuracy of the proposed method at different radii for models trained with different noise scales on the CIFAR10 dataset. The training of each model is the same as in the previous section. During certification, the number

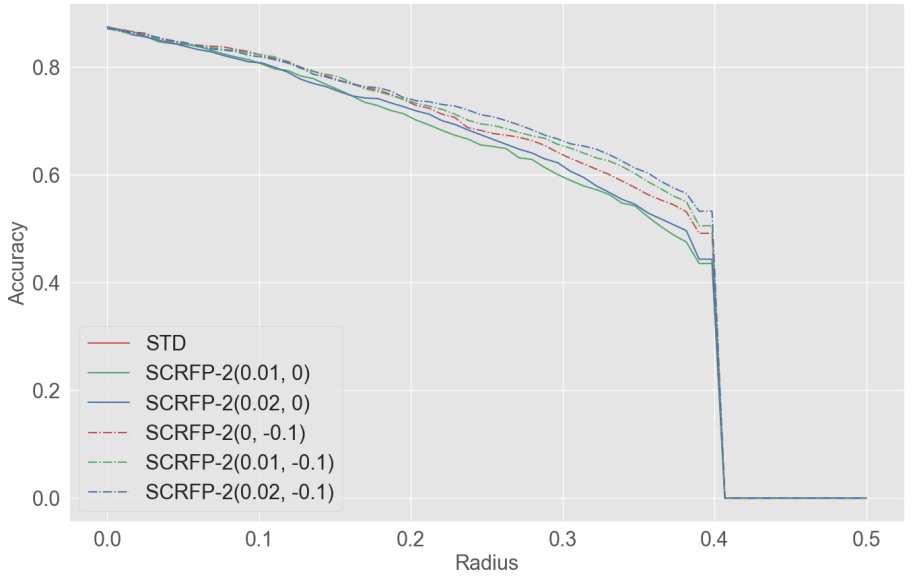


Figure 6.8: Certified accuracy under different radii on CIFAR10 dataset with different noise sizes of $\sigma = 0.125$.

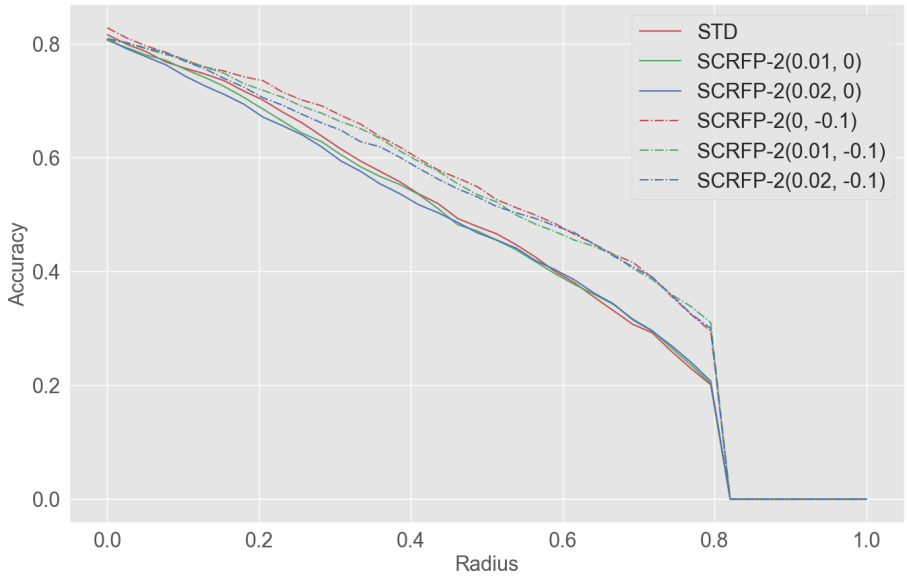


Figure 6.9: Certified accuracy under different radii on CIFAR10 dataset with different noise sizes of $\sigma = 0.25$.

of samples is set to $N = 10000$, with a failure probability $\alpha = 0.001$, following previous works.

Given $\sigma = 0.125$, Figure 6.8 shows that SCRFP-2 has comparable performance

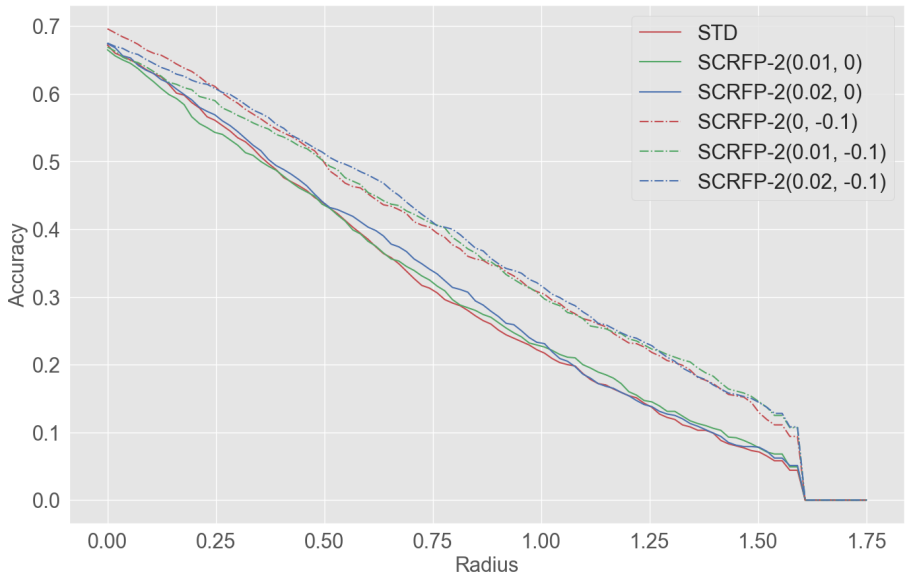


Figure 6.10: Certified accuracy under different radii on CIFAR10 dataset with different noise sizes of $\sigma = 0.50$.

Table 6.2: Certified robust accuracy for Benchmark and SCRFP-2 on CIFAR10.

	Clean	0.25	0.5	0.75	1.00	1.25	1.5
Smooth Classifier[241]	87.1	64.5	47.2	31.0	21.8	14.0	7.1
SCRFP-2(0.01, 0.00)	*87.4*	65.2	46.5	32.3	22.6	14.6	7.7
SCRFP-2(0.02, 0.00)	87.3	67.2	46.2	33.9	23.1	13.9	7.8
SCRFP-2(0.00, 0.10)	87.0	67.9	*53.9*	39.8	30.7	22.4	12.9
SCRFP-2(0.01, 0.10)	87.1	69.3	53.2	40.7	30.3	22.7	*14.5*
SCRFP-2(0.02, 0.10)	87.2	*70.9*	52.5	*40.9*	*31.5*	*23.2*	14.4

with the benchmark when the radius is close to 0, while gradually outperforming it as the radius increases. Modifying the float path during training boosts the accuracy of the model at larger radii by around 3% to 5% (red lines versus others). On the other hand, repressing the float path during prediction improves performance by around 5% compared to the benchmarks (solid lines versus dashed lines). Figures 6.9 and 6.10 present the certified accuracy for models with noise scales of 0.25 and

0.50, respectively. The results show that the effect of modifying the float path during training is less pronounced as the noise scale increases. However, repressing the float path during prediction still provides a significant boost in certified accuracy given the same radius.

Table 6.4 reports the certified accuracy at different radii. SCRFP-2 slightly boosts the benchmark model when the models are trained with refactored float paths ($\eta_1 \neq 0$) but without the prediction step ($\eta_2 = 0$). The improvement in accuracy varies from 0.3% to 2.7%. When the prediction step is introduced into the model ($\eta_2 \neq 0$), a significant improvement can be observed for all radii ¿ 0.25. For example, SCRFP-2(0.02, 0.10) outperforms the benchmark model by 9.9% at a radius of $r = 0.75$ and 9.7% at $r = 1.00$.

6.5.3 ImageNet

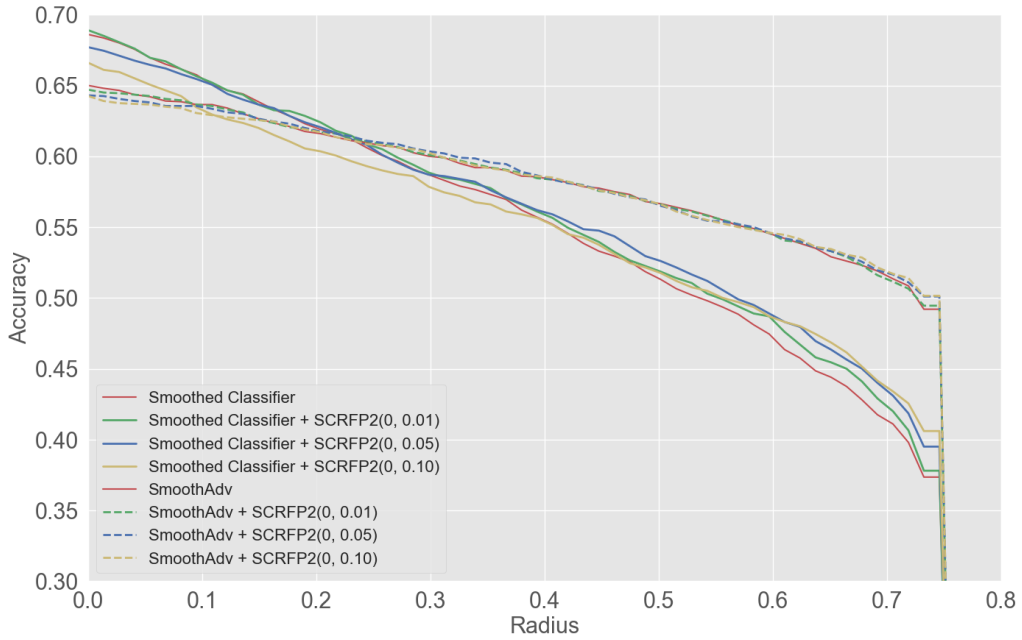


Figure 6.11: Certified accuracy under different radii on ImageNet dataset with different noise sizes of $\sigma = 0.25$.

As Theorem 10 suggests, the prediction step of SCRFP-2 can be applied to

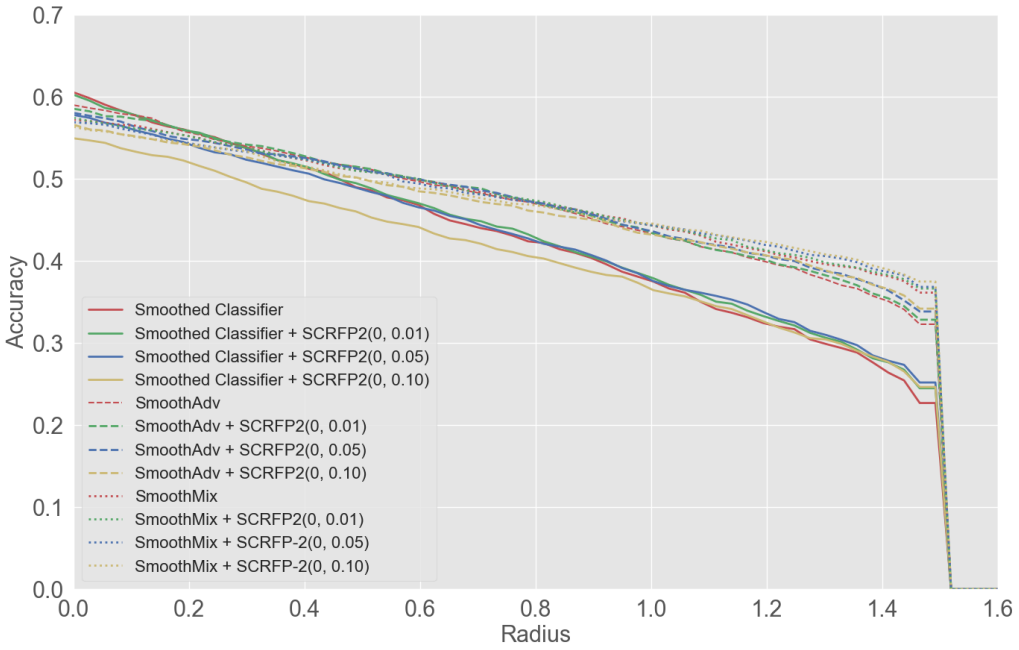


Figure 6.12: Certified accuracy under different radii on ImageNet dataset with different noise sizes of $\sigma = 0.50$.

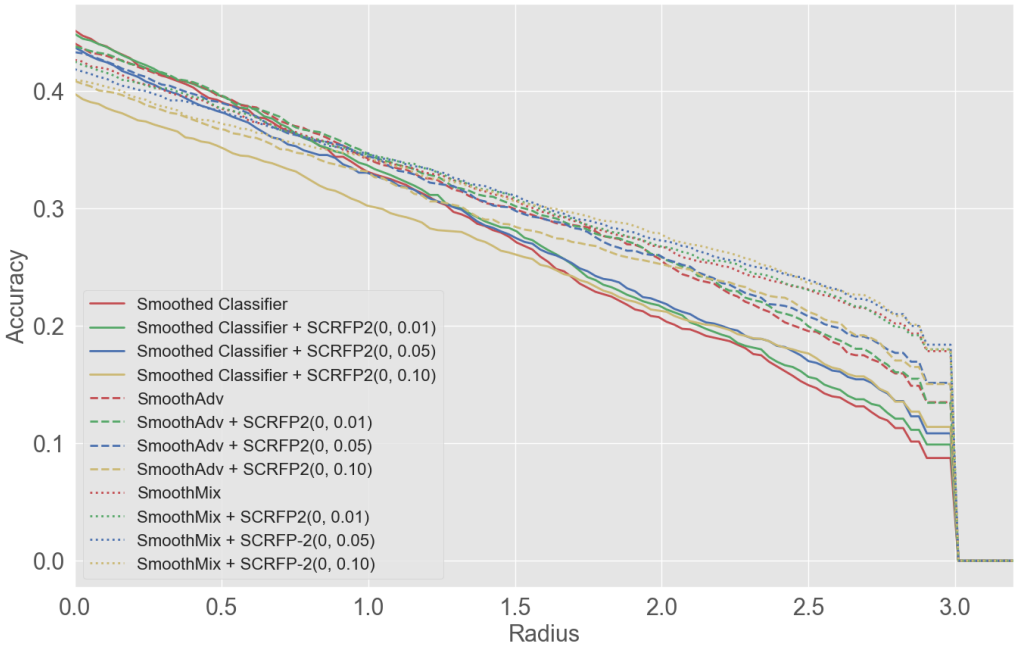


Figure 6.13: Certified accuracy under different radii on ImageNet dataset with different noise sizes of $\sigma = 1.00$.

Table 6.3: Comparison of average certified radius(ACR) between benchmarks and SCRFP-2 on ImageNet dataset.

	Smooth Classifier			SmoothAdv			SmoothMix	
σ	0.25	0.5	1.00	0.25	0.5	1.00	0.5	1.00
Benchmark	0.414	0.644	0.810	0.437	0.705	0.888	0.713	0.916
SCRFP-2(0, 0.01)	0.418	*0.650*	*0.826*	*0.438*	*0.707*	*0.895*	*0.714*	0.919
SCRFP-2(0, 0.05)	*0.419*	0.644	0.824	*0.438*	*0.707*	0.893	0.713	*0.924*
SCRFP-2(0, 0.10)	0.413	0.615	0.778	0.437	0.697	0.861	0.710	0.917

Table 6.4: Comparison of certified test accuracy between benchmark models and SCRFP-2 at different radii on ImageNet.

	0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0	2.25	2.5	2.75	3.0	Clean
Consistency*[245]	\	50	\	44	\	34	\	24	\	21	\	17	55
MACER*[255]	\	57	\	37	\	29	\	25	\	18	\	14	68
Smooth Classifer[241]	60.5	51.5	43.4	37.8	31.7	27.5	23.4	20.8	18.7	15.2	12.2	8.8	68.6
+ SCRFP2(0, 0.01)	60.7	51.9	44.1	38.1	32.1	28.2	24.3	21.8	19.1	16.1	13.2	9.9	*68.9*
+ SCRFP2(0, 0.05)	60.5	52.6	43.5	38.0	32.6	27.7	24.6	22.1	19.8	17.2	14.6	10.8	67.7
+ SCRFP2(0, 0.10)	59.2	51.9	41.3	36.9	31.3	26.2	23.8	21.3	19.7	17.6	14.6	11.4	66.6
SmoothAdv	*61.0*	*56.7*	49.2	43.5	39.1	32.3	28.4	25.8	22.8	19.7	16.9	13.5	65.0
+ SCRFP2(0, 0.01)	*61.0*	56.6	49.5	43.7	39.2	32.9	28.3	26.0	23.0	20.2	17.1	13.5	64.7
+ SCRFP2(0, 0.05)	*61.0*	56.5	*50.1*	43.7	40.0	33.9	28.1	26.1	23.5	20.9	18.4	15.2	64.3
+ SCRFP2(0, 0.10)	60.9	*56.7*	*50.1*	43.2	40.2	34.2	27.0	25.3	23.7	21.3	18.6	15.0	64.2
SmoothMix[246]	54.4	51.0	47.8	44.5	40.5	36.1	28.6	26.8	25.0	23.2	21.1	17.8	57.2
+ SCRFP2(0, 0.01)	54.4	50.9	47.9	44.4	40.8	36.6	28.7	26.8	25.4	23.2	20.8	18.0	57.5
+ SCRFP2(0, 0.05)	53.8	51.1	47.7	44.5	41.2	36.9	29.0	27.3	25.6	*24.0*	21.6	*18.4*	56.9
+ SCRFP2(0, 0.10)	53.8	50.0	47.1	*44.6*	*41.6*	*37.5*	*29.5*	*27.9*	*26.2*	23.8	*21.9*	18.0	56.4

* Reported experiment results from original paper.

any arbitrary random algorithm, provided that the sampled noise is directionally irrelevant. In this section, SCRFP-2 is applied to state-of-the-art smoothing algorithms on the ImageNet dataset. The training of the benchmark models is the same as reported in the original works. The certified radius is estimated with a sample size of $N = 10000$ and a failure probability of $\alpha = 0.001$.

6.5.3.1 Accuracy

Figures 6.11 to 6.13 compare benchmark models with SCRFP-2 (red lines versus others) at different noise scales. To evaluate the effect of the repression factor η_2, the certified radius is estimated for $\eta_2 = 0.01$, 0.05, and 0.10. The results show that SCRFP-2 is able to improve robust accuracy under different noise scales at certain radii. In particular, the boost provided by SCRFP-2 increases as the radius increases. Table 6.4 presents the certified robust accuracy of benchmark models and SCRFP-2. Without changing the training methods, SCRFP-2 is able to provide an extra 1% to 2% robust accuracy based on various state-of-the-art models on the ImageNet dataset. Moreover, SCRFP-2 is able to offer a trade-off between model performance and robustness by adjusting the repression factor η_2. For each of the models, SCRFP-2 with $\eta_2 = 0.01$ slightly boosts the accuracy of the benchmark model, while SCRFP-2 with $\eta_2 = 0.10$ provides much stronger robustness as the radius increases.

6.5.3.2 ACR

To provide a comprehensive comparison between SCRFP-2 and the benchmarks, Table 6.3 reports the *averaged certified accuracy* (ACR):

$$\text{ACR} := \frac{1}{\mathcal{D}_{\text{test}}} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}} \text{CR}(g,\sigma,\mathbf{x}) \cdot \mathbf{1}_{\hat{g}(\mathbf{x})=y}, \tag{6.49}$$

where $\text{CR}(f,\sigma,\mathbf{x})$ is the certified radius of input $\mathbf{x}$ under classifier g. By assigning a value of 0 to incorrect predictions and averaging the certified radii across the dataset, ACR (Average Certified Radius) is an accurate measure that takes the

trade-off between accuracy and robustness into account. Improvement of ACR is relatively difficult to achieve, especially at $\sigma = 0.25$ and 0.50. For example, the latest algorithm, SmoothMix, provides an additional 0.008 at $\sigma = 0.5$ compared to SmoothAdv, which improves the ACR of the benchmark model by 0.23, despite having higher accuracy at different radii. Without additional training, SCRFP-2(0, 0.01) and SCRFP-2(0, 0.05) exhibit better performance compared to other baselines. The most significant difference appears at $\sigma = 1.00$, where SCRFP-2(0, 0.05) improves the ACR of the Smooth Classifier from 0.810 to 0.826 and SmoothMix from 0.916 to 0.924.

6.5.4 Complete Experiment Results

ables 6.5 and 6.6 present the complete experimental results of SCRFP-2. On the CIFAR10 dataset, models are trained and evaluated with noise levels of $\sigma = 0.125$, 0.25, and 0.5. On the ImageNet dataset, the models are trained and evaluated with noise levels of $\sigma = 0.25$, 0.5, and 1.

Following previous works, Tables 6.2 and 6.4 summarize the complete results and present the highest certifiable accuracy for each algorithm at different radii. For completeness of the discussion, the experimental results on the CIFAR10 and ImageNet datasets are presented in this section.

6.6 Chapter Summary

This section discusses the robustness of neural networks using the proposed framework in Chapter 4. To provide a straightforward analysis, the investigation in this chapter is based on neural networks with piecewise linear activation functions.

According to Definition 1, the input space is partitioned into numerous subspaces based on the status of neurons in that space. Moreover, by merging activation regions into a larger subspace, the stability of neurons can be described using Definition 4. Inspired by the observation that the fixed neurons in a region $\mathcal{R}$

Table 6.5: Certified robust accuracy for Benchmark and SCRFP-2 on CIFAR10.

	Clean	0.25	0.5	0.75	1.00	1.25	1.5
Smooth Classifier	87.1	64.5					
SCRFP-2(0.01, 0)	*87.4*	65.2					
SCRFP-2(0.02, 0)	87.3	67.2					
SCRFP-2(0, 0.10)	87.0	67.9					
SCRFP-2(0.01, 0.10)	87.1	69.3					
SCRFP-2(0.02, 0.10)	87.2	*70.9*					
Smooth Classifier	81.6	66.9	47.2	25.9			
SCRFP-2(1, 0)	80.6	65.0	46.5	26.5			
SCRFP-2(0.02, 0)	80.8	64.1	46.2	26.9			
SCRFP-2(0, 0.10)	82.8	70.4	*53.9*	35.6			
SCRFP-2(0.01, 0.10)	80.7	69.4	53.2	35.9			
SCRFP-2(0.02, 0.10)	80.9	68.0	52.5	35.8			
Smooth Classifier	67.2	56.1	43.9	31.0	21.8	14.0	7.1
SCRFP-2(0.01, 0)	66.5	54.3	43.7	32.3	22.6	14.6	7.7
SCRFP-2(0.02, 0)	67.4	56.9	43.9	33.9	23.1	13.9	7.8
SCRFP-2(0, 0.10)	69.6	60.7	50.0	39.8	30.7	22.4	12.9
SCRFP-2(0.01, 0.10)	66.9	58.9	50.0	40.7	30.3	22.7	14.5
SCRFP-2(0.02, 0.10)	67.5	60.8	51.0	*40.9*	*31.5*	*23.2*	*14.4*

remain unchanged for any inputs, Chapter 6.2 decomposes the computational graph of the neural network. The variation in prediction between an input and its noised counterpart can also be written as a summation of a linear function and a non-linear function. Chapter 6.3 investigates the properties of these two parts and shows that unsecured data can be categorized into *Lipschitz vulnerable* and *float vulnerable*.

To demonstrate the application of the study, Chapter 6.4 further discusses the

Table 6.6: Comparison of certified test accuracy between benchmark models and SCRFP-2 with different η_2 on ImageNet at different noise scales

σ		0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0	2.25	2.5	2.75	3.0	Clean
0.25	Smooth Classifer[241]	60.5	51.5	37.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	68.6
	+ SCRFP2(0, 0.01)	60.7	51.9	37.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	*68.9*
	+ SCRFP2(0, 0.05)	60.5	52.6	39.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	67.7
	+ SCRFP2(0, 0.10)	59.2	51.9	40.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	66.6
	SmoothAdv[247]	*61.0*	*56.7*	49.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	65.0
	+ SCRFP2(0, 0.01)	*61.0*	56.6	49.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	64.7
	+ SCRFP2(0, 0.05)	*61.0*	56.5	*50.1*	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	64.3
	+ SCRFP2(0, 0.10)	60.9	*56.7*	*50.1*	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	64.2
0.50	Smooth Classifier	55.0	48.9	43.4	37.8	31.7	22.7	0.0	0.0	0.0	0.0	0.0	0.0	60.6
	+ SCRFP2(0, 0.01)	54.9	49.3	44.1	38.1	32.1	24.5	0.0	0.0	0.0	0.0	0.0	0.0	60.2
	+ SCRFP2(0, 0.05)	53.1	48.7	43.5	38.0	32.6	25.2	0.0	0.0	0.0	0.0	0.0	0.0	57.8
	+ SCRFP2(0, 0.10)	50.9	45.8	41.3	36.9	31.3	24.6	0.0	0.0	0.0	0.0	0.0	0.0	54.9
	SmoothAdv	54.7	51.3	47.7	43.5	39.1	32.3	0.0	0.0	0.0	0.0	0.0	0.0	59.0
	+ SCRFP2(0, 0.01)	54.8	51.4	48.0	43.7	39.2	32.9	0.0	0.0	0.0	0.0	0.0	0.0	58.6
	+ SCRFP2(0, 0.05)	54.4	51.2	48.2	43.7	40.0	33.9	0.0	0.0	0.0	0.0	0.0	0.0	58.1
	+ SCRFP2(0, 0.10)	53.4	50.0	46.9	43.2	40.2	34.2	0.0	0.0	0.0	0.0	0.0	0.0	56.6
	SmoothMix[246]	54.4	51.0	47.8	44.5	40.5	36.1	0.0	0.0	0.0	0.0	0.0	0.0	57.2
	+ SCRFP2(0, 0.01)	54.4	50.9	47.9	44.4	40.8	36.6	0.0	0.0	0.0	0.0	0.0	0.0	57.5
	+ SCRFP2(0, 0.05)	53.8	51.1	47.7	44.5	41.2	36.9	0.0	0.0	0.0	0.0	0.0	0.0	56.9
	+ SCRFP2(0, 0.10)	53.8	50.0	47.1	*44.6*	*41.6*	*37.5*	0.0	0.0	0.0	0.0	0.0	0.0	56.4
1.00	Smooth Classifier	41.8	39.1	36.4	33.1	30.4	27.5	23.4	20.8	18.7	15.2	12.2	8.8	45.1
	+ SCRFP2(0, 0.01)	42.1	39.5	36.6	33.7	30.9	28.2	24.3	21.8	19.1	16.1	13.2	9.9	44.9
	+ SCRFP2(0, 0.05)	40.8	38.2	35.4	33.1	30.6	27.7	24.6	22.1	19.8	17.2	14.6	10.8	43.7
	+ SCRFP2(0, 0.10)	37.4	35.2	33.2	30.2	28.1	26.2	23.8	21.3	19.7	17.6	14.6	11.4	39.8
	SmoothAdv	41.8	39.6	37.1	34.3	32.2	30.1	28.4	25.8	22.8	19.7	16.9	13.5	44.0
	+ SCRFP2(0, 0.01)	41.9	39.6	37.0	34.8	32.5	30.2	28.3	26.0	23.0	20.2	17.1	13.5	43.8
	+ SCRFP2(0, 0.05)	41.1	39.1	36.4	34.4	32.0	29.9	28.1	26.1	23.5	20.9	18.4	15.2	43.3
	+ SCRFP2(0, 0.10)	38.8	36.8	34.8	33.1	30.6	28.5	27.0	25.3	23.7	21.3	18.6	15.0	40.8
	SmoothMix	40.5	38.6	36.6	34.2	32.9	30.9	28.6	26.8	25.0	23.2	21.1	17.8	42.6
	+ SCRFP2(0, 0.01)	40.5	38.6	36.5	34.6	33.1	31.1	28.7	26.8	25.4	23.2	20.8	18.0	42.5
	+ SCRFP2(0, 0.05)	40.0	38.4	36.4	34.8	33.1	31.1	29.0	27.3	25.6	*24.0*	21.6	*18.4*	41.9
	+ SCRFP2(0, 0.10)	39.1	37.3	35.5	34.4	32.5	30.8	*29.5*	*27.9*	*26.2*	23.8	*21.9*	18.0	40.9

connection between the two types of vulnerability and shows that the essence of a smoothed classifier is to reduce sudden changes from the float path by averaging the predictions surrounding the data. To further boost the performance of the

smoothed classifier, this chapter introduces a smoothed classifier with reformed float paths in dual directions (SCRFP-2) that manipulates the float paths during both the training and prediction stages. The experimental results in Chapter 6.5 show that the proposed method can further increase the robustness of neural networks.

Chapter 7

Conclusion and Future Work

This PhD dissertation presents a thorough investigation of the interpretability of feedforward neural network from activation function perspective. As a conclusion, this chapter first summarizes the research undertaken in 7.1. It is followed by a discussion of potential future works in Chapter 7.2.

7.1 Summary of Work

The theoretical and empirical results of this work are presented mostly in Chapter 4 to Chapter 6. Inspired by the observation that neural networks with different activation functions have significant performance gap, this work provide insights on the interpretability of deep feedforward neural network with a focus on the activation functions.

7.1.1 Analytic Tools

Chapter 4 introduces a framework that developed on the basis of existing literatures. The cornerstone of this framework is *generalized activation pattern*. The innovation of the proposed definition are:

- It uses a set of pre-defined breakpoints to portion arbitrary activation functions instead, which extends the research of activation function beyond piece-wise

linear function.

- It introduces an indexed family to identify the neurons during the investigation, which enables a unit-level analysis of the model performance.

With the benefits above, the *generalized activation pattern* extend the research from a single activation region to a larger subspace in the input domain. With indices, the neurons are then categorized into *float neuron* and *fixed neuron* according to their stability in a region. Moreover, for completeness of the framework, this chapter also introduces an *incomplete activation pattern* and illustrates the geometric intuition about proposed concepts, which builds a connection between the framework in this dissertation and existing literatures.

Another contribution of chapter 4 is the discussion of the computational path of a feedforward neural network. Given a neural network with piece-wise linear activation function, the neural network is linear within each of the activation regions. As the feedforward network is a directed acyclic graph, this mapping can be decomposed into sum of paths. With the proposed framework, this property can be generalized to large subspaces by identifying the stability of neurons. This leads to the definition of *fixed path* and *float path* that describes the stability of components of the mapping function.

7.1.2 Identifying the Issue

To answer introductory question that how activation functions affect the performance of model, Chapter 3 presents experiments to verify the observation with issues that discussed by previous works. Several of the topics are covered for comparison of the models:

- It first presents empirical results that verify there exists performance gap between neural networks with different activation function.

- It then compares the weights, gradients as well as the gradient to weight

ration on different model structures. It is found that the saturated activation functions have unstable learning during the training.

- To understand the training of models, it further provides a theoretical analysis of the learning dynamic of the models. The results show that the learning of models with certain activation functions are more likely to be affected by the weights.

- At last, a toy model is demonstrated to better understand the training of those models. It is found that many neurons are firing same signal regardless of the input. In particular, the gradient of those parameters are non-zero and changes rapidly, while those neurons still fails to provide insights on the prediction.

Chapter 3 studies the performance gap of neural networks step-by-step, and locates an undocumented issue, which is referred as to *dying neuron issue*. Different from the gradient vanishing issue, the parameters unstable gradients during the training, while the output of neurons are insensitive to the inputs.

7.1.3 Interpretability of Network and Applications

The interpretability of neural network is studied from two aspects. The first part investigates the expressive ability of neural networks, with a focus of exploring the *dying neuron issue*, which is presented on Chapter 5. The second part studies the robustness of neural network and connects it with the stability of neurons, which is discussed by Chapter 6.

Based on the results from Chapter 3 and the analytic tool presented by Chapter 4, Chapter 5 explores the expressive ability of neural networks using the generalized activation pattern discussed in Chapter 4.2 under a larger scope. With a focus of understanding the *dying neuron issue* caused by different activation functions, it introduces two metrics to evaluate the severity the issue. The *pattern similarity* compares the similarity of activation pattern of a model on given dataset, which provides a general picture of the behavior of neurons in the model. The *neuron*

entropy metric evaluates the neuron level volatility of model on a dataset. This is helpful in identifying the dead neurons and understand the inner endogenous properties of a model at different layers.

Chapter 6 discusses the model robustness by decomposing the computational graph of neural network. This chapter continues the discussion of computational graph of neural network from Chapter 6.1. In particular, given neural network with piece-wise linear activation functions, the mapping within each activation region is linear and can be written as sum of the paths in a neural network. With the framework proposed in Chapter 4, this computational graph can be generalized to larger space by categorizing the paths into *fixed paths* and *float paths*. The stability of two kinds of paths are investigated. Depending on how those two paths affect the prediction of unsecured data, those data are then categorized into *Lipschitz vulnerable* and *float vulnerable*. It is shown that robustness training can reduce the Lipschitz vulnerable, while smoothed classifier is addressing the float vulnerable by averaging the prediction.

This dissertation also shows that the investigation of model stability can not only provide insights on the explaining neural network, but also helpful in developing downstream applications. In particular, this work introduces two algorithms. The neural entropy pruning (NEP) method is proposed based on the discussion of model expressive ability in Chapter 5. It aims to remove the unimportant parameters of the models to reduce the scale of neural network. The smoothed classifier with reformed float path in dual direction (SCRFP-2) aims to improve the robustness of smoothed classifier. It is inspired by the observation that the instability of neural network is largely resulted by the float path therefore manipulate the value on float path during training and prediction to achieve a better performance.

7.2 Future Works

This work introduces a framework to explore the interpretability of feedforward neural networks. It is shown that this framework is helpful in studying the expressive ability and robustness of deep learning models. Beyond the scope and contribution of this work, there are other unaddressed research topics for the explainable AI. This section discusses the potential future works from several aspects.

7.2.1 Convolutional Model

This research primarily explores the expressive capability and robustness of feedforward neural networks, examining the role of activation functions while deliberately omitting details about the linear operators involved. Feedforward networks, typically characterized by their sequence of layers and activation functions, do not explicitly consider the spatial or structural aspects of input data. However, the convolutional neural network (CNN), a variant designed specifically for processing data with inherent spatial hierarchies (such as images), incorporates convolutional operators that significantly impact performance. These convolutional operators rely on parameters such as kernel size, stride, and padding, each playing a crucial role in determining how the network interprets and processes input data.

In CNNs, the kernel size affects the receptive field of the convolution operation, determining the extent of the input region considered for each output pixel. A larger kernel size increases the receptive field, potentially capturing more global features but at the cost of computational efficiency and possible over-smoothing of features. Conversely, a smaller kernel size focuses on local features, which can enhance model sensitivity but may miss broader contextual information. The stride, or the step size with which the kernel moves across the input, also influences the output size and the level of detail captured; larger strides produce smaller output dimensions, which can reduce spatial resolution and detail.

This research proposes that by systematically varying these parameters and

observing the resultant effects on model performance and robustness, significant insights can be gained into the operational dynamics of CNNs. Furthermore, such an investigation can extend beyond performance metrics to address aspects of neural network explainability. By understanding how changes in convolutional parameters affect layer activations and feature maps, researchers can gain clearer insights into the internal representations and decision-making processes within CNNs. This approach not only enhances our understanding of neural network architectures and their functionalities but also contributes to the broader discourse on making machine learning models more interpretable and trustworthy.

7.2.2 Other Deep Learning Models

Apart from feedforward neural network, there are other deep learning models that have been applied to various fields. For example, the recurrent neural networks are widely used in the natural language processing, while the computational graph has recurrent structure that does not satisfy the prerequisite of this framework. However, the recurrent neural network still uses activation functions to provide non-linearity and approximating objective functions. This means that it can also be analyzed from the activation function perspective. In particular, by further generalizing the framework proposed in this dissertation and removes the constraint on directed acyclic graph, the computational graph and stability of RNN can be explored.

Another mainstream topic in the research of deep learning is the generative AI. The research of deep learning based generative model starts from the study of generative adversarial networks, which is further outperformed by the diffusion models. Similar to the models discussed in this work, model of the generative models are also feedforward neural networks but with different objective functions and training algorithm. This means that further investigation of the learning of generative models with the proposed framework can also shed lights on the interpretability of generative models.

7.2.3 Bounding the Computational Paths for Continuous Functions

In Chapter 6, the discussion of model robustness is limited for neural networks with piece-wise linear activation function. However, as stated in the discussion, it can also be generalized to continuous activation function by adding a bound for each of the partition of the activation function.

The decomposition of the piece-wise linear models are developed from the additive of activation function. This idea can also be adopted to the continuous activation functions. In particular, by splitting the input domain of continuous activation function according to the function properties, the output and derivative of the region is bounded. Therefore, within each of the activation region, the upper bound and lower bound of the mapping function can be formulated. This means that the robustness of models with continuous functions can be investigated by generalizing Lemma 11 and Theorem 8.

Appendix A

Notations

This chatper summarizes the notations and lists the definitions and concepts proposed in this dissertation. In particular,

- Section A.1 introduces the analytical tools used in this dissertation, including activation patterns, activation regions, fixed / float neurons, and paths.
- Section A.2 is presented as a preliminary for the investigation of expressive ability by listing and explaining the metrics and notations used in Chapter 5.
- Similarly, Section A.3 defines the concepts used in Chapter 6, which discusses the robustness of neural networks.

A.1 Activation Pattern, Region and Path

This section summarizes the notations for the proposed concepts in this dissertation. Given a neural network $\mathcal{N}$ with activation function π, let Γ be a set of breakpoints of size q that divide the domain of the activation function into $q+1$ intervals. Each breakpoint and interval is denoted as $\gamma_i, i = 1, \ldots, q$, and $U_i, i = 0, \ldots, q$, respectively.

Section 4.2 introduces the activation pattern and activation region. A neuron in the neural network $\mathcal{N}$ is indexed by a tuple (i, j), where i is the layer and j is

the neuron number, meaning that neuron (i, j) is the j-th neuron in layer i. The collection of indices of neuron in network is denoted as $\mathcal{I}$.

$$\mathcal{I} := \{(i, j) | i = 1, 2, \ldots, d-1, j \in [n_d]\}, \tag{A.1}$$

where n_d is the dimension of layer d, $[n_d] = \{1, 2, \ldots, n_d\}$.

A pattern of a neuron is a label for the neuron (i, j) indicating which interval the pre-activation value $z^{(i)}j(\mathbf{x}; \theta)$ falls into, and is denoted as $\hat{a}^{(i)}j(\mathbf{x}; \theta, \pi, \Gamma)$. The indexed collection of patterns for all the neurons in the network forms the activation pattern of the neural network, denoted as $\mathcal{A}$. Given an activation pattern $\mathcal{A}$, $\mathcal{R}(\mathcal{A}; \theta, \pi, \Gamma)$ is the corresponding activation region, as defined in Definition 1.

The activation region $\mathcal{R}(\mathcal{A}; \theta, \pi, \Gamma)$ of an activation pattern $\mathcal{A}$ can be viewed as an operator that maps the index family to a subspace in $\mathbb{R}^n$. Similarly, given input $\mathbf{x} \in \mathbb{R}^n$, the pattern of each neuron and the overall activation pattern of the data are denoted as $\hat{a}^{(i)}j(\mathbf{x}; \theta, \pi, \Gamma)$ and $\hat{\mathcal{A}}(\mathbf{x}; \theta, \pi, \Gamma)$, respectively. These operators map data in $\mathbb{R}^n$ to an index family. A bent hyperplane is denoted as $Hijk(\theta)$ to represent the case where the pre-activation value of neuron (i, j) equals the k-th breakpoint:

$$H_{ijk}(\theta) = \{\mathbf{x} | z_j^{(i)}(\mathbf{x}) = \gamma_k\}. \tag{A.2}$$

Section 4.3 generalizes the activation pattern into a larger subspace by defining incomplete activation patterns and fixed/float neurons. As defined in Definition 3, given a subset of the indices $\mathcal{I}^c$, an incomplete activation pattern is defined as $\mathcal{A}_{\mathcal{I}^c}$, where only the neurons in $\mathcal{I}^c$ are assigned a sign. The corresponding region in the input space is referred to as the incomplete activation region $\mathcal{R}(\mathcal{A}_{\mathcal{I}^c})$. Given a subspace $\mathcal{R} \subset \mathbb{R}^n$, $\mathcal{I}^X(\mathcal{R})$ and $\mathcal{I}^F(\mathcal{R})$ represent the fixed neurons and float neurons, respectively, as defined in Definition 4, to describe the stability of neuron activation patterns.

Section 6.1 introduces the concepts of fixed paths and float paths. A path ζ is defined as a sequenced collection of neurons that starts from the first layer and extends to the $(d-1)$-th layer, containing a single neuron from each layer, as defined in Definition 7. The value of a path is denoted as $\zeta(\mathbf{x}, \mathcal{A})$. Given a subspace $\mathcal{R}$, a

Activation Pattern	
Γ	A set of breakpoints that separates the activation function.
$\mathcal{A}$	An activation pattern of $\mathcal{N}$.
$\mathcal{A}_{\mathcal{I}^c}$	An incomplete activation pattern: $\{a_j^{(i)} \mid a_j^{(i)} \in \mathcal{A}, (i,j) \in \mathcal{I}^C\}$.
$\mathcal{R}(\mathcal{A}; \theta, \pi, \Gamma)$	An activation region defined by pattern $\mathcal{A}$.
$\hat{\mathcal{A}}(\mathbf{x}; \theta, \pi, \Gamma)$	The activation pattern of $\mathbf{x}$.
$\hat{a}_j^{(i)}(x; \theta, \pi, \Gamma)$	The pattern of neuron (i,j) for input x.
H_{ijk}	A bent-hyperplane defined by neuron (i,j) and k-th breakpoint.
$\mathcal{I}$	Collection of indices of neuron in network $\mathcal{N}$.
$\mathcal{I}^C$	An incomplete activation pattern $\mathcal{I}^C \subset \mathcal{I}$.
$\mathcal{I}^X(\mathcal{R})$	Collection of indices of fixed neuron of network $\mathcal{N}$ in region R.
$\mathcal{I}^T(\mathcal{R})$	Collection of indices of float neuron of network $\mathcal{N}$ in region $\mathcal{R}$.
ζ	A path of network $\mathcal{N}$.
$d_{\zeta_m}^{(m)}$	The slop of activation pattern $\hat{a}_j^{(i)}$.
$\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R})$	Value of fixed path in region $\mathcal{R}$ given input x and pattern$\mathcal{A}$.
$\mathcal{Z}^T(x, \mathcal{A}; \mathcal{R})$	Value of float path in region $\mathcal{R}$ given input x and pattern$\mathcal{A}$.

Table A.1: Notations of Activation Pattern, Region and Path.

path ζ is considered a fixed path if every neuron on the path is a fixed neuron in $\mathcal{R}$; otherwise, it is a float path. The summed values of all fixed paths and float paths are denoted as $\mathcal{Z}^I(\mathbf{x}, \mathcal{A}; \mathcal{R})$ and $\mathcal{Z}^T(\mathbf{x}, \mathcal{A}; \mathcal{R})$, respectively, which are introduced in the discussion of Definition 8. The notations for activation patterns, activation regions, and activation paths are presented in Table A.1.

A.2 Expressive Ability

Chapter 5 investigates the model's expressive ability by comparing neural networks with different activation functions, as well as the dying neuron issue identified in

Chapter 3. Given a neural network $\mathcal{N}$ and inputs $\mathbf{x}, \mathbf{x}' \in \mathbb{R}$, the distance between $\mathbf{x}$ and $\mathbf{x}'$ is denoted as $D(\mathbf{x}, \mathbf{x}')$, while the prediction difference between $\mathbf{x}$ and $\mathbf{x}'$ is denoted as $\delta(\mathbf{x}, \mathbf{x}')$ with:

$$\delta(\mathbf{x}, \mathbf{x}') = D(f(\mathbf{x}), f(\mathbf{x}')).$$

For such a pair of data, $\overline{\mathbf{x}\mathbf{x}'}$ is the segment connecting $\mathbf{x}$ and $\mathbf{x}'$, and $\mathbf{TD}(\overline{\mathbf{x}\mathbf{x}'})$ is the transition density introduced in previous work [182]. $\mathbf{TD}(\overline{\mathbf{x}\mathbf{x}'})$ measures how many transitions occur as $\mathbf{x}$ moves to $\mathbf{x}'$, where a transition is counted by a transition indicator $\mathbf{Tra}(\mathbf{x}_i, \mathbf{x}_{i+1})$ to show whether the segment $\overline{\mathbf{x}_i\mathbf{x}_{i+1}}$ between $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ crosses a bent hyperplane or not.

Pattern similarity is a metric introduced in Definition 5 to study the performance gap between neural networks with different activation functions. Given $\mathbf{x}$ and $\mathbf{x}'$, $\mathbf{PS}(\mathbf{x}, \mathbf{x}'; \pi, \boldsymbol{\theta}, \Gamma)$ measures the similarity between the activation patterns of $\mathbf{x}$ and $\mathbf{x}'$. Given a data distribution $\mathbb{D}$, $\mathbf{PS}(\mathbb{D}, \pi, \boldsymbol{\theta}, \Gamma)$ is the expected pattern similarity for $\mathbb{D}$, while $\mathbf{PS}(\mathbb{D}, \lambda)$ refers to the distribution of pattern similarity introduced to study how the pattern similarity changes across different thresholds λ.

The second part of Chapter 5 delves into the expressive ability of neural networks, with a focus on further explaining the dying neuron issue for a single network. Definition 6 introduces a metric called neuron entropy, which measures the stability of neurons. The entropy of the j-th neuron in layer i is denoted as $\mathcal{E}_j^{(i)}$. The notations used in this work for studying the expressive ability of neural networks are shown in Table A.2.

A.3 Robustness

Another research thread in the interpretability of neural networks is model robustness, which is discussed in Chapter 6 of this dissertation. Given input $\mathbf{x} \in \mathbb{R}^n$, a sphere centered at $\mathbf{x}$ with radius r, measured under the p-norm, is denoted as $B_p(\mathbf{x}, r)$. Let $\mathbf{x}' = \mathbf{x} + \boldsymbol{\epsilon}$ be a noised sample of $\mathbf{x}$, where $\boldsymbol{\epsilon}$ is a perturbation. This chapter studies the decomposition of the computational graph on $\mathbf{x}$ and $\mathbf{x}'$. Similar

Expressive Ability	
$D(\mathbf{x}, \mathbf{x}')$	The distance between $\mathbf{x}$ and $\mathbf{x}$: $\|\mathbf{x} - \mathbf{x}'\|$.
$\delta(\mathbf{x}, \mathbf{x}')$	The prediction difference between $\mathbf{x}$ and $\mathbf{x}'$: $\|f(\mathbf{x}) - f(\mathbf{x}')\|$.
$\overline{\mathbf{xx}'}$	Segment connecting $\mathbf{x}$ and $\mathbf{x}'$.
$\mathbf{Tra}(x_i, x_{i+1})$	Transition indicator to show whether $\overline{\mathbf{x}_i\mathbf{x}_{i+1}}$ crosses a bent-hyperplane.
$TD(\overline{\mathbf{xx}'})$	Transition density of $\overline{\mathbf{xx}'}$.
$\mathbf{PS}(\mathbf{x}, \mathbf{x}'; \pi, \theta, \Gamma)$	Pattern similarity between $\mathbf{x}$ and $\mathbf{x}'$.
$\mathbf{PS}(\mathbb{D}; \pi, \boldsymbol{\theta}, \Gamma)$	Pattern similarity on data distribution $\mathbb{D}$.
$\mathbf{PS}(\mathbb{D}, \lambda; \pi, \boldsymbol{\theta}, \Gamma)$	Distribution of pattern similarity on data distribution $\mathbb{D}$.
$\mathcal{CS}^I(\mathbf{x}, \mathbf{x}')$	Cosine similarity between fixed path of $\mathbf{x}$ and $\mathbf{x}'$.
$\mathcal{CS}^T(\mathbf{x}, \mathbf{x}')$	Cosine similarity between float path of $\mathbf{x}$ and $\mathbf{x}'$.
$\mathcal{E}_j^{(i)}(\mathcal{N}, \mathbb{D}_x)$	Entropy of neuron (i, j) given network $\mathcal{N}$ on distribution $\mathbb{D}_x$.

Table A.2: Notations for Investigation of Model Expressive Ability.

to the definition of prediction difference, after decomposing the prediction for a neural network with piecewise linear activation functions, the differences between $\mathbf{x}$ and $\mathbf{x}'$ on the fixed paths and float paths are denoted as $\delta^I(\mathbf{x}, \mathbf{x}')$ and $\delta^T(\mathbf{x}, \mathbf{x}')$, respectively. The cosine similarity between the fixed path and float path of $\mathbf{x}$ and $\mathbf{x}'$ is then denoted as $\mathcal{CS}^I$ and $\mathcal{CS}^T$.

The investigation into the robustness of neural networks leads to a downstream application proposed based on the randomized smoothing algorithm. Given a network $\mathcal{N}$ with mapping function f, g is a smoothed classifier built on f. In the case that randomness is generated by a normal distribution, the standard deviation of the noise is denoted as σ. Therefore, the noise distribution with zero mean and independent correlation is $N(0, \sigma^2 I)$. For such a randomized classifier, $\underline{p_A}$ and $\overline{p_B}$ are the lower and upper bounds for $g'_{m \neq y}(\mathbf{x})$, and $\Phi^{-1}(\cdot)$ is the inverse of the normal distribution used to compute the certifiable radius of the smoothed classifier.

Robustness	
$B_p(\mathbf{x}, r)$	A sphere that centered at $\mathbf{x}$ with radius r that measured under p-norm.
$\delta^I(\mathbf{x}, \mathbf{x}')$	Difference between fixed path of $\mathbf{x}$ and $\mathbf{x}' : \mathcal{Z}^I(\mathbf{x}) - \mathcal{Z}^I(\mathbf{x}')$
$\delta^T(\mathbf{x}, \mathbf{x}')$	Difference between float path of $\mathbf{x}$ and $\mathbf{x}' : [f(\mathbf{x}) - f(\mathbf{x}')] - \delta^I(\mathbf{x}, \mathbf{x}')$.
g	Smoothed Classifier.
σ	Variance of Gaussian Noise for smoothed classifier.
$\underline{p_A}$	The lower bound for $g_{m \neq y}(\mathbf{x})$ given smoothed classifier g.
$\overline{p_B}$	The upper bound for $g_{m \neq y}(\mathbf{x})$ given smoothed classifier g.
$\Phi^{-1}(\cdot)$	Inverse of normal distribution.
ϵ	Perturbation added to the input $\mathbf{x}$
ξ	Unit Vector with norm 1.
η_1	Amplification factor of SCRFP-2 during training.
η_2	Repression factor of SCRFP-2 during predicting.

Table A.3: Notations for Investigation of Model Robustness.

Bibliography

[1] Z. Jiang, "On explaining neural network robustness with activation path," in *The Eleventh International Conference on Learning Representations*, 2023.

[2] Z. Jiang, Y. Wang, C.-T. Li, P. Angelov, and R. Jiang, "Delve into activations: Towards understanding dying neuron," *IEEE Transactions on Artificial Intelligence*, 2022.

[3] Z. Jiang, P. L. Chazot, and R. Jiang, "Review on social behavior analysis of laboratory animals: From methodologies to applications," in *Recent advances in AI-enabled automated medical diagnosis*, CRC Press, 2022, pp. 110–122.

[4] Z. Jiang, P. L. Chazot, M. E. Celebi, D. Crookes, and R. Jiang, "Social behavioral phenotyping of drosophila with a 2d–3d hybrid cnn framework," *IEEE Access*, vol. 7, pp. 67 972–67 982, 2019.

[5] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *stat*, vol. 1050, p. 2, 2017.

[6] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang, "A survey on neural network interpretability," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 5, pp. 726–742, 2021.

[7] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, IEEE, 2018, pp. 80–89.

[8] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature machine intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5353–5360.

[13] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[16] S. Park, C. Yun, J. Lee, and J. Shin, "Minimum width for universal approximation," in *International Conference on Learning Representations*, 2020.

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[19] B. Hanin and D. Rolnick, "Complexity of linear regions in deep networks," in *International Conference on Machine Learning*, 2019, pp. 2596–2604.

[20] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023. DOI: 10.1109/JPROC.2023.3238524.

[21] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digital Signal Processing*, vol. 126, p. 103 514, 2022.

[22] L. Alzubaidi, J. Zhang, A. J. Humaidi, *et al.*, "Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[23] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[24] M. Treviso, J.-U. Lee, T. Ji, *et al.*, "Efficient methods for natural language processing: A survey," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 826–860, 2023.

[25] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.

[26] S. Bond-Taylor, A. Leach, Y. Long, and C. G. Willcocks, "Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models," *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[27] L. Yang, Z. Zhang, Y. Song, *et al.*, "Diffusion models: A comprehensive survey of methods and applications," *arXiv preprint arXiv:2209.00796*, 2022.

[28] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001.

[29] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[30] L. Ciampiconi, A. Elwood, M. Leonardi, A. Mohamed, and A. Rozza, "A survey and taxonomy of loss functions in machine learning," *arXiv preprint arXiv:2301.05579*, 2023.

[31] R.-Y. Sun, "Optimization for Deep Learning: An Overview," *Journal of the Operations Research Society of China*, vol. 8, no. 2, pp. 249–294, Jun. 2020, ISSN: 2194-668X, 2194-6698. DOI: 10.1007/s40305-020-00309-6. (visited on 09/17/2023).

[32] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. [Online]. Available: probml.ai.

[33] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[34] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd. Springer, 2009.

[35] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 Workshop on Learning for Text Categorization*, 1998, pp. 41–48.

[36] T. Joachims, "A probabilistic analysis of the rocchio algorithm with tfidf for text categorization," in *Proceedings of the 14th International Conference on Machine Learning (ICML)*, 1997, pp. 143–151.

[37] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes - which naive bayes?" In *Third Conference on Email and Anti-Spam (CEAS)*, 2006.

[38] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, 2002, pp. 79–86.

[39] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[40] C. M. Bishop, "Pattern recognition and machine learning (information science and statistics)," *Springer New York*, 2007.

[41] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[42] J. A. Bilmes, "A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models," in *International Computer Science Institute*, vol. 4, 1998, p. 126.

[43] R. Adams and L. Bischof, "Seeded region growing," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, vol. 16, 1994, pp. 641–647.

[44] L. R. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.

[45] D. A. Reynolds, "Gaussian mixture models," *Encyclopedia of Biometrics*, pp. 659–663, 2009.

[46] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[47] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, "Support vector clustering," *Journal of machine learning research*, vol. 2, no. Dec, pp. 125–137, 2001.

[48] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, *A practical guide to support vector classification*, 2003.

[49] H. Bhavsar and M. H. Panchal, "A review on support vector machine for data classification," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 1, no. 10, pp. 185–189, 2012.

[50] K.-B. Duan and S. S. Keerthi, "Which is the best multiclass svm method? an empirical study," in *International workshop on multiple classifier systems*, Springer, 2005, pp. 278–285.

[51] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[52] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *Journal of Machine Learning Research*, vol. 5, pp. 101–141, 2004.

[53] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[54] J. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin dags for multiclass classification," in *Advances in Neural Information Processing Systems*, 2000, pp. 547–553.

[55] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *Journal of Artificial Intelligence Research*, vol. 2, pp. 263–286, 1995.

[56] S. T. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," in *Proceedings of the Seventh International Conference on Information and Knowledge Management (CIKM)*, 1998, pp. 148–155.

[57] W. S. Noble, "What is a support vector machine?" *Nature Biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[58] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[59] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[60] B. Mor, S. Garhwal, and A. Kumar, "A systematic review of hidden markov models and their applications," *Archives of computational methods in engineering*, vol. 28, pp. 1429–1448, 2021.

[61] A. Betkowska, K. Shinoda, and S. Furui, "Robust speech recognition using factorial hmms for home environments," *EURASIP Journal on advances in signal processing*, vol. 2007, pp. 1–9, 2007.

[62] J.-L. Durrieu and J.-P. Thiran, "Source/filter factorial hidden markov model, with application to pitch and formant tracking," *IEEE transactions on audio, speech, and language processing*, vol. 21, no. 12, pp. 2541–2553, 2013.

[63] N. Raman and S. J. Maybank, "Activity recognition using a supervised non-parametric hierarchical hmm," *Neurocomputing*, vol. 199, pp. 163–177, 2016.

[64] S. Karaman, J. Benois-Pineau, V. Dovgalecs, *et al.*, "Hierarchical hidden markov model in detecting activities of daily living in wearable videos for studies of dementia," *Multimedia tools and applications*, vol. 69, pp. 743–771, 2014.

[65] H. Xiong and R. Mamon, "A self-updating model driven by a higher-order hidden markov chain for temperature dynamics," *Journal of Computational Science*, vol. 17, pp. 47–61, 2016.

[66] A. Ozerov, C. Févotte, and M. Charbit, "Factorial scaled hidden markov model for polyphonic audio representation and source separation," in *2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, IEEE, 2009, pp. 121–124.

[67] S. B. Kotsiantis, "Decision trees: A recent overview," *Artificial Intelligence Review*, vol. 39, pp. 261–283, 2013.

[68] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[69] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[70] J. Gehrke, R. Ramakrishnan, and V. Ganti, "Rainforest—a framework for fast decision tree construction of large datasets," *Data Mining and Knowledge Discovery*, vol. 4, pp. 127–162, 2000.

[71] K.-j. Kim, "Financial time series forecasting using support vector machines," *Neurocomputing*, vol. 55, no. 1-2, pp. 307–319, 2003.

[72] Y. Cao, Y. Li, S. Coleman, A. Belatreche, and T. M. McGinnity, "Adaptive hidden markov model with anomaly states for price manipulation detection," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 2, pp. 318–330, 2014.

[73] V. Podgorelec, P. Kokol, B. Stiglic, and I. Rozman, "Decision trees: An overview and their use in medicine," *Journal of medical systems*, vol. 26, pp. 445–463, 2002.

[74] A. Awaysheh, J. Wilcke, F. Elvinger, L. Rees, W. Fan, and K. L. Zimmerman, "Review of medical decision support and machine-learning methods," *Veterinary pathology*, vol. 56, no. 4, pp. 512–525, 2019.

[75] C. P. Janssen, L. N. Boyle, A. L. Kun, W. Ju, and L. L. Chuang, "A hidden markov framework to capture human–machine interaction in automated vehicles," *International Journal of Human–Computer Interaction*, vol. 35, no. 11, pp. 947–955, 2019.

[76] M. Kabra, A. A. Robie, M. Rivera-Alba, S. Branson, and K. Branson, "Jaaba: Interactive machine learning for automatic annotation of animal behavior," *Nature methods*, vol. 10, no. 1, pp. 64–67, 2013.

[77] E. Eyjolfsdottir, S. Branson, X. P. Burgos-Artizzu, *et al.*, "Detecting social actions of fruit flies," in *European Conference on Computer Vision*, Springer, 2014, pp. 772–787.

[78] H. Jhuang, E. Garrote, X. Yu, *et al.*, "Automated home-cage behavioural phenotyping of mice," *Nature communications*, vol. 1, p. 68, 2010.

[79] M. Vrigkas, C. Nikou, and I. A. Kakadiaris, "A review of human activity recognition methods," *Frontiers in Robotics and AI*, vol. 2, p. 28, 2015.

[80] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[81] M. Papadonikolakis and C.-S. Bouganis, "A novel fpga-based svm classifier," in *2010 International Conference on Field-Programmable Technology*, IEEE, 2010, pp. 283–286.

[82] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[83] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2020.

[84] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[85] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: Knn-based ensemble of classifiers," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2016, pp. 1192–1195.

[86] Y. Tang, *Deep Learning using Linear Support Vector Machines*, Feb. 2015. arXiv: `1306.0239 [cs, stat]`. (visited on 09/17/2023).

[87] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[88] G. E. Hinton, "Neural network architectures for artificial intelligence," American Association for Artificial Intelligence, 1988.

[89] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[90] T. Lin, M. Maire, S. J. Belongie, *et al.*, "Microsoft COCO: common objects in context," *CoRR*, 2014. arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312.

[91] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[92] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[93] Y. Tian, R. Sukthankar, and M. Shah, "Spatiotemporal deformable part models for action detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2013.

[94] F. Gu, M.-H. Chung, M. Chignell, S. Valaee, B. Zhou, and X. Liu, "A survey on deep learning for human activity recognition," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–34, 2021.

[95] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: State of the art, current trends and challenges," *Multimedia tools and applications*, vol. 82, no. 3, pp. 3713–3744, 2023.

[96] M.-T. Luong, I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba, "Addressing the rare word problem in neural machine translation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, pp. 11–19.

[97] A. Killawala, I. Khokhlov, and L. Reznik, "Computational Intelligence Framework for Automatic Quiz Question Generation," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Rio de Janeiro: IEEE, Jul. 2018, pp. 1–8, ISBN: 978-1-5090-6020-7. DOI: 10.1109/FUZZ-IEEE.2018.8491624. (visited on 09/17/2023).

[98] G. Wiese, D. Weissenborn, and M. Neves, "Neural domain adaptation for biomedical question answering," in *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, 2017, pp. 281–289.

[99] M. V. Mäntylä, D. Graziotin, and M. Kuutila, "The evolution of sentiment analysis—A review of research topics, venues, and top cited papers," *Computer Science Review*, vol. 27, pp. 16–32, Feb. 2018, ISSN: 15740137. DOI: 10.1016/j.cosrev.2017.10.002. (visited on 09/17/2023).

[100] K. L. Tan, C. P. Lee, K. S. M. Anbananthen, and K. M. Lim, "Roberta-lstm: A hybrid model for sentiment analysis with transformer and recurrent neural network," *IEEE Access*, vol. 10, pp. 21 517–21 525, 2022.

[101] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[102] H. Wang, Y. Zhang, X. Yu, *et al.*, "An overview of image caption generation methods," *Computational intelligence and neuroscience*, vol. 2020, 2020.

[103] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[104] N. M. S. Surameery and M. Y. Shakor, "Use chat gpt to solve programming bugs," *International Journal of Information Technology & Computer Engineering (IJITC) ISSN: 2455-5290*, vol. 3, no. 01, pp. 17–22, 2023.

[105] E. L. Hill-Yardin, M. R. Hutchinson, R. Laycock, and S. J. Spencer, "A chat (gpt) about the future of scientific publishing," *Brain Behav Immun*, vol. 110, pp. 152–154, 2023.

[106] A. Lopez-Lira and Y. Tang, "Can chatgpt forecast stock price movements? return predictability and large language models," *Return Predictability and Large Language Models (April 6, 2023)*, 2023.

[107] S. S. Biswas, "Role of chat gpt in public health," *Annals of biomedical engineering*, vol. 51, no. 5, pp. 868–869, 2023.

[108] G. Sebastian, "Do chatgpt and other ai chatbots pose a cybersecurity risk?: An exploratory study," *International Journal of Security and Privacy in Pervasive Computing (IJSPPC)*, vol. 15, no. 1, pp. 1–11, 2023.

[109] K. Fuchs, "Exploring the opportunities and challenges of nlp models in higher education: Is chat gpt a blessing or a curse?" In *Frontiers in Education*, Frontiers, vol. 8, 2023, p. 1 166 682.

[110] J. Achiam, S. Adler, S. Agarwal, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[111] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[112] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[113] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[114] V. Gulshan, L. Peng, M. Coram, *et al.*, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *Jama*, vol. 316, no. 22, pp. 2402–2410, 2016.

[115] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[116] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[117] K. Cho, B. van Merrienboer, C. Gulcehre, *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[118] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[119] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," *2013 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 6645–6649, 2013.

[120] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[121] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, pp. 303–338, 2010.

[122] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer Vision and Pattern Recognition Workshop*, 2004.

[123] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[124] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[125] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[126] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[127] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[128] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[129] A. Howard, M. Sandler, G. Chu, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[130] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[131] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[132] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[133] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[134] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6154–6162.

[135] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[136] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[137] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[138] P. Rajpurkar, J. Irvin, K. Zhu, *et al.*, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," *arXiv preprint arXiv:1711.05225*, 2017.

[139] A. Esteva, B. Kuprel, R. A. Novoa, *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[140] V. Buhrmester, D. Münch, and M. Arens, "Analysis of explainers of black box deep neural networks for computer vision: A survey," *Machine Learning and Knowledge Extraction*, vol. 3, no. 4, pp. 966–989, 2021.

[141] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," *Advances in neural information processing systems*, vol. 30, 2017.

[142] V. Maiorov and A. Pinkus, "Lower bounds for approximation by mlp neural networks," *Neurocomputing*, vol. 25, no. 1-3, pp. 81–91, 1999.

[143] N. J. Guliyev and V. E. Ismailov, "On the approximation by single hidden layer feedforward neural networks with fixed weights," *Neural Networks*, vol. 98, pp. 296–304, 2018.

[144] D.-X. Zhou, "Universality of deep convolutional neural networks," *Applied and computational harmonic analysis*, vol. 48, no. 2, pp. 787–794, 2020.

[145] R. Brüel Gabrielsson, "Universal function approximation on graphs," *Advances in neural information processing systems*, vol. 33, pp. 19 762–19 772, 2020.

[146] A. Chouldechova and T. Hastie, "Generalized additive model selection," *arXiv preprint arXiv:1506.03850*, 2015.

[147] M. Craven and J. Shavlik, "Extracting tree-structured representations of trained networks," *Advances in neural information processing systems*, vol. 8, 1995.

[148] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[149] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin, "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation," *journal of Computational and Graphical Statistics*, vol. 24, no. 1, pp. 44–65, 2015.

[150] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, Springer, 2014, pp. 818–833.

[151] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR (workshop track)*, 2015.

[152] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness," in *International Conference on Learning Representations*, 2018.

[153] K. Xu, J. Ba, R. Kiros, *et al.*, "Neural image caption generation with visual attention," in *Proc. ICML*, 2015, pp. 2048–2057.

[154] H. K. B. Babiker and R. Goebel, "An introduction to deep visual explanation," *arXiv preprint arXiv:1711.09482*, 2017.

[155] M. T. Ribeiro, S. Singh, and C. 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*, 2016, pp. 1135–1144.

[156] U. von Luxburg and O. Bousquet, "Distance-based classification with lipschitz functions.," *J. Mach. Learn. Res.*, vol. 5, no. Jun, pp. 669–695, 2004.

[157] K. Scaman and A. Virmaux, "Lipschitz regularity of deep neural networks: Analysis and efficient estimation," *arXiv preprint arXiv:1805.10965*, 2018.

[158] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation," in *NIPS*, 2017.

[159] B. Neyshabur, S. Bhojanapalli, D. Mcallester, and N. Srebro, "Exploring generalization in deep learning," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5947–5956, 2017.

[160] Y. Tsuzuku, I. Sato, and M. Sugiyama, "Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[161] J. Peck, J. Roels, B. Goossens, and Y. Saeys, "Lower bounds on the robustness to adversarial perturbations," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[162] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[163] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," in *Conference on learning theory*, PMLR, 2016, pp. 698–728.

[164] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, "Exponential expressivity in deep neural networks through transient chaos," *Advances in neural information processing systems*, vol. 29, pp. 3360–3368, 2016.

[165] Q. Li, S. Haque, C. Anil, J. Lucas, R. B. Grosse, and J.-H. Jacobsen, "Preventing gradient attenuation in lipschitz constrained convolutional networks," *Advances in neural information processing systems*, vol. 32, pp. 15 390–15 402, 2019.

[166] H. Sedghi, V. Gupta, and P. M. Long, "The singular values of convolutional layers," in *International Conference on Learning Representations*, 2018.

[167] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, "Parseval networks: Improving robustness to adversarial examples," in *International Conference on Machine Learning*, PMLR, 2017, pp. 854–863.

[168] R. Pascanu, G. Montufar, and Y. Bengio, "On the number of response regions of deep feed forward networks with piece-wise linear activations," *arXiv preprint arXiv:1312.6098*, 2013.

[169] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in neural information processing systems*, 2014, pp. 2924–2932.

[170] B. Hanin and D. Rolnick, "Deep relu networks have surprisingly few activation patterns," in *Advances in Neural Information Processing Systems*, 2019, pp. 361–370.

[171] M. Jordan, J. Lewis, and A. G. Dimakis, "Provable certificates for adversarial examples: Fitting a ball in the union of polytopes," *Advances in neural information processing systems*, vol. 32, 2019.

[172] H. Zhang, S. Wang, K. Xu, *et al.*, "A branch and bound framework for stronger adversarial attacks of relu networks," in *International Conference on Machine Learning*, PMLR, 2022, pp. 26 591–26 604.

[173] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, 2022.

[174] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *et al.*, *Gradient flow in recurrent nets: The difficulty of learning long-term dependencies*, 2001.

[175] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, Pmlr, 2013, pp. 1310–1318.

[176] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[177] C. Szegedy, W. Zaremba, I. Sutskever, *et al.*, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[178] Y. Luo, X. Boix, G. Roig, T. Poggio, and Q. Zhao, "Foveation-based mechanisms alleviate adversarial examples," *arXiv preprint arXiv:1511.06292*, 2015.

[179] A. Rozsa, M. Gunther, and T. E. Boult, "Towards robust deep neural networks with bang," *arXiv preprint arXiv:1612.00138*, 2016.

[180] J. Ba and R. Caruana, "Do deep nets really need to be deep?" *Advances in neural information processing systems*, vol. 27, 2014.

[181] J. Lee, L. Xiao, S. Schoenholz, *et al.*, "Wide neural networks of any depth evolve as linear models under gradient descent," *Advances in neural information processing systems*, vol. 32, pp. 8572–8583, 2019.

[182] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivity and generalization in neural networks: An empirical study," *arXiv preprint arXiv:1802.08760*, 2018.

[183] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, "Spectrally-normalized margin bounds for neural networks," *Advances in Neural Information Processing Systems*, vol. 30, pp. 6240–6249, 2017.

[184] B. Neyshabur, S. Bhojanapalli, and N. Srebro, "A pac-bayesian approach to spectrally-normalized margin bounds for neural networks," in *International Conference on Learning Representations*, 2018.

[185] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," in *Conference on learning theory*, PMLR, 2020, pp. 2306–2327.

[186] M. Telgarsky, "Benefits of depth in neural networks," in *Conference on learning theory*, PMLR, 2016, pp. 1517–1539.

[187] H. W. Lin, M. Tegmark, and D. Rolnick, "Why does deep and cheap learning work so well?" *Journal of Statistical Physics*, vol. 168, no. 6, pp. 1223–1247, 2017.

[188] W. Liu, Y.-M. Zhang, X. Li, *et al.*, "Deep hyperspherical learning," *Advances in neural information processing systems*, vol. 30, 2017.

[189] M. Böhle, M. Fritz, and B. Schiele, "B-cos networks: Alignment is all we need for interpretability," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 329–10 338.

[190] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6541–6549.

[191] C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su, "This looks like that: Deep learning for interpretable image recognition," *Advances in neural information processing systems*, vol. 32, 2019.

[192] P. Angelov and E. Soares, "Towards explainable deep neural networks (xdnn)," *Neural Networks*, vol. 130, pp. 185–194, 2020.

[193] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.

[194] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[195] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*, IEEE, 1993, pp. 293–299.

[196] T.-J. Yang, A. Howard, B. Chen, *et al.*, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

[197] T.-J. Yang, Y.-L. Liao, and V. Sze, "Netadaptv2: Efficient neural architecture search with fast super-network training and architecture optimization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2402–2411.

[198] Z. Guo, X. Zhang, H. Mu, *et al.*, "Single path one-shot neural architecture search with uniform sampling," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI*, 2020, pp. 544–560.

[199] G. Bender, H. Liu, B. Chen, *et al.*, "Can weight sharing outperform random architecture search? an investigation with tunas," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 323–14 332.

[200] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[201] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[202] Y. Xu, L. Xie, X. Zhang, *et al.*, "Pc-darts: Partial channel connections for memory-efficient architecture search," in *International Conference on Learning Representations*, 2019.

[203] J. Mei, Y. Li, X. Lian, *et al.*, "Atomnas: Fine-grained end-to-end neural architecture search," in *International Conference on Learning Representations*, 2020.

[204] A. Rosenfeld and J. K. Tsotsos, "Incremental learning through deep adaptation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 3, pp. 651–663, 2018.

[205] A. Dubey, M. Chatterjee, and N. Ahuja, "Coreset-based neural network compression," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 454–470.

[206] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, 2015, pp. 1135–1143.

[207] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2017.

[208] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[209] N. Lee, T. Ajanthan, and P. Torr, "Snip: Single-shot network pruning based on connection sensitivity," in *International Conference on Learning Representations*, 2019.

[210] R. Yu, A. Li, C.-F. Chen, *et al.*, "Nisp: Pruning networks using neuron importance score propagation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, 2018, pp. 9194–9203.

[211] B. Biggio, I. Corona, D. Maiorca, *et al.*, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*, Springer, 2013, pp. 387–402.

[212] W. Ruan, X. Yi, and X. Huang, "Adversarial robustness of deep learning: Theory, algorithms, and applications," in *Proceedings of the 30th ACM international conference on information & knowledge management*, 2021, pp. 4866–4869.

[213] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *stat*, vol. 1050, p. 20, 2015.

[214] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[215] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," in *International Conference on Learning Representations*, 2019.

[216] J. Wang, "Adversarial examples in physical world.," in *IJCAI*, 2021, pp. 4925–4926.

[217] Y. Dong, F. Liao, T. Pang, *et al.*, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9185–9193.

[218] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *International Conference on Machine Learning*, PMLR, 2020, pp. 2206–2216.

[219] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *International Conference on Learning Representations*, 2016.

[220] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, "Detecting adversarial samples from artifacts," *arXiv preprint arXiv:1703.00410*, 2017.

[221] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE symposium on security and privacy (SP)*, IEEE, 2016, pp. 582–597.

[222] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense against adversarial attacks using high-level representation guided denoiser," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1778–1787.

[223] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," *arXiv preprint arXiv:1607.04311*, 2016.

[224] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 3–14.

[225] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," *arXiv preprint arXiv:2001.03994*, 2020.

[226] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[227] A. Shafahi, M. Najibi, A. Ghiasi, *et al.*, "Adversarial training for free!" *arXiv preprint arXiv:1904.12843*, 2019.

[228] D. Zhang, T. Zhang, Y. Lu, Z. Zhu, and B. Dong, "You only propagate once: Accelerating adversarial training via maximal principle," *arXiv preprint arXiv:1905.00877*, 2019.

[229] C. Zhu, Y. Cheng, Z. Gan, S. Sun, T. Goldstein, and J. Liu, "Freelb: Enhanced adversarial training for natural language understanding," *arXiv preprint arXiv:1909.11764*, 2019.

[230] M. Andriushchenko and N. Flammarion, "Understanding and improving fast adversarial training," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 048–16 059, 2020.

[231] B. Li, S. Wang, S. Jana, and L. Carin, "Towards understanding fast adversarial training," *arXiv preprint arXiv:2006.03089*, 2020.

[232] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[233] A. Kurakin, I. Goodfellow, S. Bengio, *et al.*, *Adversarial examples in the physical world*, 2016.

[234] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *International conference on machine learning*, PMLR, 2018, pp. 274–283.

[235] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference on Machine Learning*, PMLR, 2018, pp. 5286–5295.

[236] E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter, "Scaling provable adversarial defenses," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[237] S. Lee, J. Lee, and S. Park, "Lipschitz-certifiable training with a tight outer bound," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 891–16 902, 2020.

[238] L. Weng, H. Zhang, H. Chen, *et al.*, "Towards fast computation of certified robustness for relu networks," in *International Conference on Machine Learning*, PMLR, 2018, pp. 5276–5285.

[239] H. Zhang, H. Chen, C. Xiao, *et al.*, "Towards stable and efficient training of verifiably robust neural networks," *arXiv preprint arXiv:1906.06316*, 2019.

[240] Y. Huang, H. Zhang, Y. Shi, J. Z. Kolter, and A. Anandkumar, "Training certifiably robust neural networks with efficient local lipschitz bounds," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 745–22 757, 2021.

[241] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *International Conference on Machine Learning*, PMLR, 2019, pp. 1310–1320.

[242] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 656–672.

[243] A. Levine, S. Singla, and S. Feizi, "Certifiably robust interpretation in deep learning," *arXiv preprint arXiv:1905.12105*, 2019.

[244] B. Li, C. Chen, W. Wang, and L. Carin, "Certified adversarial robustness with additive noise," *Advances in neural information processing systems*, vol. 32, 2019.

[245] J. Jeong and J. Shin, "Consistency regularization for certified robustness of smoothed classifiers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 558–10 570, 2020.

[246] J. Jeong, S. Park, M. Kim, H.-C. Lee, D.-G. Kim, and J. Shin, "Smoothmix: Training confidence-calibrated smoothed classifiers for certified robustness," *Advances in Neural Information Processing Systems*, vol. 34, pp. 30 153–30 168, 2021.

[247] H. Salman, J. Li, I. Razenshteyn, *et al.*, "Provably robust deep learning via adversarially trained smoothed classifiers," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[248] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *stat*, vol. 1050, p. 21, 2016.

[249] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[250] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[251] J. Von Oswald, D. Zhao, S. Kobayashi, *et al.*, "Learning where to learn: Gradient sparsity in meta and continual learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5250–5263, 2021.

[252] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, PMLR, 2015, pp. 448–456.

[253] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *international conference on machine learning*, PMLR, 2017, pp. 2847–2854.

[254] H. Federer, *Geometric measure theory*. Springer, 2014.

[255] R. Zhai, C. Dan, D. He, *et al.*, "Macer: Attack-free and scalable robust training via maximizing certified radius," *arXiv preprint arXiv:2001.02378*, 2020.