Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Qihang Zhang April 8, 2019

Differential Equation Interpretation of Deep Neural Networks

by

Qihang Zhang

Lars Ruthotto

Adviser

Department of Mathematics

Lars Ruthotto

Adviser

Yuanzhe Xi

Committee Member

David Fisher

Committee Member

2019

Differential Equation Interpretation of Deep Neural Networks

By

Qihang Zhang

Lars Ruthotto

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Abstract

Differential Equation Interpretation of Deep Neural Networks

By Qihang Zhang

Deep Neural Networks have become the state-of-the-art tools for supervised machine learning with the ability to extract features and find patterns from complicated input data. Although there were no general guidelines for how to design good architectures that generalize well to unseen data, researchers have recently proposed the connection between deep neural network structures and ordinary differential equations. This innovative approach has gained much attention and has been widely accepted. In this thesis, we firstly illustrate the continuous interpretation of deep neural networks on a concrete example. To do so, we use a modified Residual Neural Network structure, which allows us to discretize the network and the weights separately. While the step size to discretize the network is big, the network cannot classify the labels well. However, as the step size gets smaller, the network's classification performance gets better. Next, based on the continuous interpretation, we look into the stability of the forward propagation of a deep neural network. We compute the eigenvalues of the Jacobian matrix of each time point. Furthermore, we visualize the objective function of a deep neural network to explore how to modify the network structure to make the training more efficient. Last, we propose new forward propagation architecture with Runge Kutta method of order four and compare it with the forward propagation of a standard Residual Neural Network.

Differential Equation Interpretation of Deep Neural Networks

By

Qihang Zhang

Lars Ruthotto

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Acknowledgements

First, I would like to thank my advisor, Dr. Lars Ruthotto, for giving me this precious opportunity to work with him and introducing me to the field of deep learning. Dr. Ruthotto not only guides me patiently through my research process, but also helps me determine my next goal after graduating from college. I would also like to thank the other two committee members, Dr. Yuanzhe Xi and Dr. David Fisher, for participating in my thesis defense. This project would not be possible without generous support from Emory's SURE and honors programs. Finally, I would like to thank my family for supporting my studies at Emory and letting me choose my own career path.

Contents

1 Introduction **1**
1.1 Contributions and Outline 3

2 Background **5**
2.1 Theoretical Background of Deep Neural Networks 6
2.2 Network Architecture 6
2.2.1 Loss Function and Optimization Methods 9
2.2.2 The Residual Neural Network 10
2.3 Differential Equation Interpretation 11
2.4 Stability of Deep Neural Networks 13
2.4.1 Stability of ODEs 13
2.5 Runge Kutta Methods 15

3 Experiments and Results **17**
3.1 ResNets with different step sizes 21
3.2 Stability of the ResNet Forward Propagation 25
3.3 Visualizing the Loss Function 29
3.4 RK4 Forward Propagation 33

4 Summary and Conclusion **36**

5 MATLAB® Code **39**

List of Figures

3.1 Input features of the Peaks Example 19
3.2 Illustration of state and control layer 20
3.3 Classification Result 23
3.4 Eigenvalues of an 8-layer ResNet 27
3.5 Eigenvalues of a 16-layer ResNet 27
3.6 Eigenvalues of a 64-layer ResNet 28
3.7 Loss landscape of an 8-layer ResNet 31
3.8 Loss landscape of a 20-layer ResNet 32
3.9 Objective functions of RK4 and Forward Euler 35

Chapter 1

Introduction

Since the 1950s, researchers and scientists have started to explore the field of machine learning. In 1957, the idea of deep neural networks (DNNs) was first introduced to the world by Frank Rosenblatt as a computational model that would be able to make decisions based on given information just like a human's brain. However, even though the concept has been proposed for a long time, the progress of the study had been rather slow due to the lack of data collection and low computational power. Fortunately, things have changed in 1989 when Yann LeCun developed the convolutional neural networks (CNNs) along with the backpropagation algorithm. It was a great progress in the development of DNNs as the network was able to recognize handwritten digits in checks and zip codes with a high accuracy. As a special type of DNNs, CNNs utilize convolutional operators to reduce the computational cost while preserving the spatial structure of the input data. Nowadays, CNNs are

widely applied in real-world tasks including image classification and speech recognition. With the eminent performance of CNNs, the development of DNNs has regained the public's attention.

Along with the improvement of computational power, DNNs have now become one of the most popular tools for supervised machine learning, and scholars have been working on constructing deeper network structures with the approach to add more layers to the network to retrieve better network performance. Through the past decade, several groundbreaking network structures have been invented, including the AlexNet [8] with five convolutional layers and GoogleNet [12] with twenty-two layers. However, network performance does not improve solely through adding more layers to the structure. Due to the notorious vanishing gradient problem [7], the performance of a network might even degrade rapidly as the network gets deeper. Meanwhile, the Residual Neural Network (ResNet), which was introduced in [7] in 2015, could unprecedentedly reach hundreds or even thousands of layers with accurate classification performance with the implementation of skip connection technique which would keep the gradient from vanishing.

Despite the outstanding performance achieved by ResNet, there still exist numerous critical issues that must be resolved. For example, the robustness of DNNs against adversarial attacks has remained one of the most concerning problems [13]. In particular, if a neural network is not stable, then some small perturbations in the input data might change the output significantly, thus lead to catastrophic consequences. For example, unstable autonomous

vehicles could possibly miss a stop sign and hit a pedestrian.

In order to avoid those fatal accidents, researchers have proposed various intriguing methods, and one especially promising approach appeared in recent years. The core idea of this approach is to consider the network structure from a mathematical angle, implementing the network structure as the discretization of ordinary differential equations (ODEs) [5].

In this thesis, several discretization methods of ODEs - including Forward Euler method and Runge Kutta method- will be introduced. An ODE discretization is a numerical method to transfer the continuous ODE into discrete counterparts and retrieve an approximate solution that is close to the true solution of the original problem [1]. In this way, researchers can apply the abundant knowledge of partial differential equations to the field of DNNs. In our study, we conduct experiments to further validate this approach and more specifically, to show how different discretization implementations will affect the training of DNNs.

1.1 Contributions and Outline

In this thesis, I provide experimental evidence for the applicability of differential equations interpretation of DNNs. In Chapter 2 I give brief background information about the neural network framework and the differential equation interpretation of the neural network. In Chapter 3 I give details about the experiments that I have done along with the results. This chapter

includes:

- Numerical experiment of the ODE interpretation of DNNs on a concrete example
- Numerical study of the stability condition of the ResNet forward propagation
- Visualization and analysis of the effect of neural network structures on the efficiency of network training
- New forward propagation algorithm implemented with RK4 method

Finally, in Chapter 4 I discuss the importance of these results to the machine learning community. The MATLAB codes written for computing the results are included in Chapter 5.

Chapter 2

Background

In order to help the readers to understand the bedrock of this thesis, which is the differential equation interpretation of deep neural networks (DNNs), we introduce related mathematical topics based on the knowledge and explanations from [5, 1, 7, **?**, 4, 6, 10, 2] in this section. First, I present the generalized mathematical description of DNNs and expand on this to the specific structure of the Residual Neural Network (ResNet). Second, I explain the main idea of this paper, which is how the ResNet structure is related to the forward Euler discretization method and thus how we can use the differential equation knowledge to interpret the networks. Finally, I provide a theoretical background of the new network structure we construct based on higher order Runge Kutta discretization methods.

2.1 Theoretical Background of Deep Neural Networks

A deep neural network (DNN) is an advanced technique for various real-world applications such as data classification and predictive analysis. Mathematically speaking, A DNN is a trainable function from an input space $X \in \mathbb{R}^n$ to an output space $Y \in \mathbb{R}^m$. It is a concatenation of many simple functions parametrized by initially randomized weights $\mathbf{w}$. Each function is called a layer of the network. The goal is to optimize the weights such that elements from the X space can be classified most accurately. In particular, we start with a group of data $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_s, \mathbf{y}_s)\}$ where for each $i \in (1, \ldots, s)$, we want the network to take $\mathbf{x}_i$ as the input feature vector and give us an accurate prediction of the class label vector $\mathbf{y}_i$ and finally be able to predict the label of unseen data from X.

2.2 Network Architecture

In this section, the ingredients of a DNN that are related to the thesis are introduced. The notations used are based on the work done in [2, 5].

A neuron is the smallest unit in a neural network. Each neuron serves as a single function which receives the output value of each neuron from the previous layer, noted as x_i, as the input values and then computes an output value a_j (also known as the activation of the unit). To achieve a_j, we first

compute z_j as the linear combination of each input x_j and the corresponding weight w_{ij} of the i^{th} output and the j^{th} input from the current layer and then add the bias b_j as

$$z_j = \sum_{i=1}^{p} w_{ij} x_i + b_j, \tag{2.1}$$

where p is the number of neurons in the previous layer. Then, we pass z_i to a non-linear activation function σ, which is applied element-wised, to obtain

$$a_j = \sigma(z_j). \tag{2.2}$$

The activation function mentioned above is crucial to network training. As σ introduces non-linearity to the network, we increase the complexity of the network, which enables us to approximate more complex functional mappings from the input data other than just affine functions. Two common activation functions are the tanh activation function

$$\sigma_{tanh}(\mathbf{Z}) = \tanh(\mathbf{Z}), \tag{2.3}$$

and the rectified linear unit (ReLU) activation function

$$\sigma_{ReLU}(\mathbf{Z}) = \max(0, \mathbf{Z}). \tag{2.4}$$

As we zoom out to a larger scope, a network layer consists of numbers of neurons based on the decision of the network designer, and the forward propagation algorithm describes how each layer passes the evaluated outputs to the next layer as the new inputs. The mathematical intuition of forward propagation is to combine the function of each neuron and vectorize across the full layer, and the formula expression of the forward propagation is

$$\mathbf{X}_{j+1} = \sigma(\mathbf{W}_j \mathbf{X}_j + \mathbf{b}_j \mathbf{e}_k). \tag{2.5}$$

To describe the forward propagation mathematically, we denote $\mathbf{X}_0 = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_s]$ as the input values, $\mathbf{W}_j$ and $\mathbf{b}_j$ as the corresponding weight matrix and the bias term of j^{th} layer, and $\mathbf{e}_k$ is a k-dimensional vector of all ones. A non-linear activation function σ is then applied to the linear combination of the inputs, weights, and the bias to generate the outputs of the current layer.

After the output of the output layer $\mathbf{X}_N$ has been computed, a hypothesis function $h(\mathbf{\Phi} \mathbf{X}_N + \boldsymbol{\mu} \mathbf{e}_k)$ is used to calculate the class label probabilities. The weight matrix associated to the hypothesis function is denoted as $\mathbf{\Phi}$ and the bias is $\boldsymbol{\mu}$ multiplied to a k-dimensional vector of all ones e_k.

There are several common choices for hypothesis functions. The logistic regression function

$$h(x) = \exp(x)/(1 + \exp(x)) \tag{2.6}$$

is often used for Bernoulli variables ($k = 1$), and the softmax function

$$h(\mathbf{X}) = \exp(\mathbf{X})/(\exp(\mathbf{X})e_s) \tag{2.7}$$

is often used for multinomial distributions ($k > 1$).

2.2.1 Loss Function and Optimization Methods

Another key component of a DNN is the loss function (or an objective function) l. As we mentioned in Section 2.2, a neural network takes a number of input feature vectors $\mathbf{x}_i$ paired with its class label vectors $\mathbf{y}_i$ to train the initially randomized weights $\mathbf{W}$ in order to predict the class label of unseen data samples. The loss function enables us to determine how well the neural network makes predictions with the trained parameters of the forward propagation $\mathbf{W}$ and $\mathbf{b}$, and the hypothesis function parameters $\mathbf{\Phi}$, and $\boldsymbol{\mu}$ [5]. In particular, we want to minimize the minimization problem, i.e.,

$$\arg\min \frac{1}{k} \sum_{i=1}^{k} l(h(\mathbf{\Phi}\mathbf{X}_N + \boldsymbol{\mu}\mathbf{e}_k), \mathbf{Y}). \tag{2.8}$$

A simple choice of the loss function l is the sum-of-square difference function $l(\mathbf{Y}, \mathbf{Y}_{\text{pred}}) = \frac{1}{2}||\mathbf{Y}_{\text{pred}} - \mathbf{Y}||_F^2$. In our experiments, the cross-entropy function is applied.

To solve problem (2.8), one of the most popular algorithms is the stochastic gradient descent (SGD) method [2], which is an iterative method that

updates all the weights based on the gradient of a randomly chosen data sample. We implement the backpropagation algorithm [9] to compute the gradients in a reverse order starting from the output layer to the input layer.

In our experiments, we applied another optimization method, which is the Gauss-Newton method.

2.2.2 The Residual Neural Network

In the past decade, one approach that has gained much attention is to increase the network depth. However, as more layers are added to the network, new problems have arisen. By simply stacking more layers, one cannot guarantee the increase of classification accuracy and would actually decrease the accuracy due to the degradation problem [7].

A Residual Neural Network (ResNet) is a special type of DNN that is largely applied in real-world tasks such as image classification and speech recognition. It was first proposed in [7] based on simple modifications to the common network structure. Unlike a common network, a ResNet utilizes shortcut connections between network layers. The theoretical motivation of the structure is easy to understand. As mentioned in [7], the principle of this network is to ensure that a model with more depth should produce no higher training error than its shallower counterpart. Assume $H(\mathbf{x})$ as an underlying mapping to be fit by a few stacked layers, where $\mathbf{x}$ denotes the inputs to the first of these layers. The authors of [7] then introduced the residual mapping

$F(\mathbf{x})$, where

$$F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}, \tag{2.9}$$

assuming $\mathbf{x}$ and $H(\mathbf{x})$ are of the same dimensions. Intuitively speaking, it is easier to optimize the residual mapping than to optimize the original desired mapping, because if $\mathbf{x}$ is optimal already, the residual term can be just driven to 0. A ResNet could construct a deeper network with the same level of accuracy, and the mathematical formula of a simplified ResNet is

$$\mathbf{X}_{j+1} = \mathbf{X}_j + \sigma(\mathbf{W}_j\mathbf{X}_j + b_j) \text{ for } j \in \{0, 1, \ldots, N\}. \tag{2.10}$$

Even though ResNets have achieved great success in the DNN field, many critical problems regarding the network structure have not yet been solved. In particular, the development of DNNs has been mostly based on historical knowledge and theoretical results with complex assumptions that are hard to interpret [10]. Thus, innovative perspectives of neural network interpretation seem necessary to the further development of DNNs.

2.3 Differential Equation Interpretation

Among all the researchers that works on exploring the features of ResNet, the authors of [5] were the first, to our knowledge, to purpose a connection

between DNN structures and continuous dynamic systems. These authors observed that the forward propagation structure of a ResNet is the same as the forward Euler discretization of an ordinary differential equation (ODE). With a minor modification to the forward propagation structure introduced in section 2.2.2, the authors generalized the forward propagation of a ResNet as

$$\mathbf{X}_{j+1} = \mathbf{X}_j + h\sigma(\mathbf{W}_j\mathbf{X}_j + b_j) \text{ for } j \in \{0, 1, \ldots, N\}. \tag{2.11}$$

Here, h is introduced to the equation that could be interpreted as the step size in a finite difference approximation of the change of $\mathbf{X}$ with respect to an artificial time variable t as

$$\frac{\mathbf{X}_{j+1} - \mathbf{X}_j}{h} = \sigma(\mathbf{W}_j\mathbf{X}_j + b_j). \tag{2.12}$$

In the original forward propagation structure, h is set to be 1. As pointed out in [6], with the interpretation of $\mathbf{W}_j\mathbf{X}_j$ as the discretization of $s * x$, when $h \to 0$, the ResNet could be interpreted as a continuous problem

$$\frac{dx}{dt}(t) = \sigma(s(t)x(t) + b(t)),\ x(0) = \mathbf{X}_0, \tag{2.13}$$

for $t \in [0, T]$, where T is the final time corresponding to the output layer.

The ODE interpretation of the ResNet has brought novel insight into

the design of network structure and potential solutions to the critical issues that have been impeded the wider application of DNNs [5]. One of the most pervasive problems encountered in network construction is the phenomenon of exploding and vanishing gradients, where vanishing gradients indicate the insensitivity of the output with respect to the input and exploding gradients show the instability of the output to the input. Furthermore, we also notice limitations of ResNets from the network structure. As we known from the forward Euler method, which is a first order method, the time step h must be kept rather small for satisfying stability. Theoretically speaking, DNNs with higher order forward propagation should give more robust performance and more accurate classifications. With the help of the continuous ODE interpretation, researchers are now able to study the stability of DNNs with the rich knowledge of ODEs. We will discuss the stability of DNNs and ODEs, and potential solutions to the problem in next section.

2.4 Stability of Deep Neural Networks

One promising approach to improve the robustness of network structures is to look into the stability of their underlying ODEs [5].

2.4.1 Stability of ODEs

Theoretically speaking, an ODE is stable if small perturbations of initial data lead to only small variations of the solution, i.e., the solution is robust to the

initial value. Based on the theory stated in [5], an ODE is stable if

$$\max(Re(\lambda(J(t)))) \leq 0, \ \forall \ t \in [0, T] \tag{2.14}$$

where $Re(\cdot)$ means the real part, $\lambda(\cdot)$ means the eigenvalues of a squared matrix, and $J(t)$ is the Jacobian matrix

$$J(t) = \text{diag}(\sigma'(W(t)x(t) + b(t)))W(t) \tag{2.15}$$

The Jacobian matrix is the matrix of all first partial derivatives of a given function, and the eigenvalues of the Jacobian matrix would tell us how much variance there is in the data in the same direction as the eigenvalue's corresponding eigenvector is pointing to. Mathematically speaking, if in a neural network, $\exists \ t \in [0, T] \ s.t. \ Re(\lambda(J(t))) > 0$, then it means that some features that are passed through the network will be amplified without an upper bound. On the other hand, a non-positive real part of an eigenvalue indicates the convergence of the systems, i.e., the ODE is stable. Nevertheless, if the real part of an eigenvalue is much smaller than zero, i.e, $Re(\lambda(J(t))) \ll 0$, then relevant information from the input data could not be preserved properly. This type of networks is defined as "lossy". However, a moderate lossy network has its own advantage while dealing with noisy input data because higher order oscillation (noise) in the data is tend to be reduced when the processed data is passing to the next layer.

In summary, the stability of the forward propagation of a DNN can be

obtained when

$$Re(\lambda(J(t))) \approx 0 \ \forall \ t \in [0, T]. \tag{2.16}$$

For the eigenvalues that have real parts that are close to zero, the feature of the data would be only amplified or shrunk in a moderate sense, i.e., even a neural network with a considerable depth would still be stable enough for effective learning.

2.5 Runge Kutta Methods

As the relationship between DNNs and dynamic systems has been explored in [5] and [6], a novel approach of using continuous dynamic systems as a tool for deep learning has gained more attention. Since a ResNet can be regarded as an approximation of a time-dependent dynamical system utilizing the forward Euler method (also known as Runge Kutta method of order 1), higher order Runge Kutta methods could also be implemented to create new network models intuitively.

The Runge Kutta Methods generate higher-order approximations of the right hand side of (2.13) by utilizing repeated function evaluations at the current step t_i for a more accurate approximation of x_{i+1} at next step t_{i+1}. In this section, we introduce the most commonly used Runge Kutta method of order 4 (RK4), i.e., the classical RK method [1]. The RK4 method uses

four explicit K stages to achieve fourth order accuracy. The formula is given as

$$K_1 = x_i \tag{2.17}$$

$$K_2 = x_i + \frac{h}{2} f(t_i, K_1) \tag{2.18}$$

$$K_3 = x_i + \frac{h}{2} f(t_{i+\frac{1}{2}}, K_2) \tag{2.19}$$

$$K_4 = x_i + h f(t_{i+\frac{1}{2}}, K_3) \tag{2.20}$$

$$x_{i+1} = x_i + \frac{h}{6}(f(t_i, K_1) + 2f(t_{i+\frac{1}{2}}, K_2) + 2f(t_{i+\frac{1}{2}}, K_3) + f(t_i, K_4)). \tag{2.21}$$

As the concept of differential equations interpretation of DNNs is accepted by more and more scholars, it is reasonable to believe that RK methods can be adopted to design effective network architectures as well. In our experiment, a new forward propagation structure based on RK4 method is introduced and the performance is compared to the forward propagation of a standard ResNet.

Chapter 3

Experiments and Results

The main purpose of conducting the first two experiments is to study the ordinary differential equation (ODE) interpretation of deep neural networks (DNNs) [5] and the stability of DNNs on a concrete example. In the third experiment, we implemented a loss function visualization method [10] and studied how different network structures affect the efficiency of training. In the last experiment, we constructed a new forward propagation based on Runge Kutta method of order four.

Data Selection

The first three experiments were conducted on the Peaks data set described in [5]. The Peaks data set is based on the peaks function in MATLAB®

$$\begin{aligned} f(x) =& 3(1-x_1)^2 \exp(-({x_1}^2) - (x_2+1)^2) - 10(x_1/5 - {x_1}^3 - {x_2}^5) \\ & \exp(-{x_1}^2 - {x_2}^2) - \frac{1}{3}\exp(-(x_1+1)^2 - {x_2}^2) \end{aligned}$$

which has two variables x_1 and x_2, and $x \in [-3,3]^2$. The peaks function is smooth but has some nonlinearities and, most importantly, non-convex level sets. The function is discretized on a regular 256×256 grid and all the points are separated into five different classes based on the corresponding function values. After the five classes are determined, the training data is then randomly sampled from each class, such that the sampled data set approximately represents the whole level sets. In the experiments, a total of 5,000 sample points were collected as training data, composed of 1,000 data points were randomly selected from each of the five classes. Figure 3.1 from [5] illustrates the data sampled for our experiments.

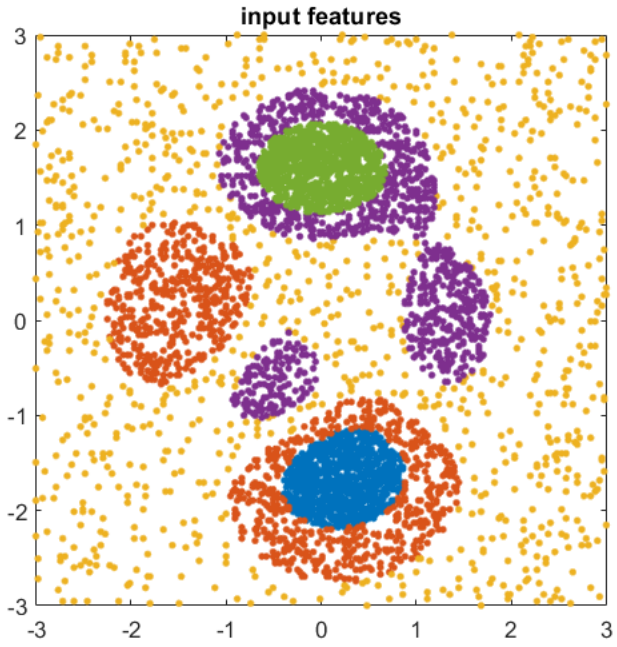

Figure 3.1: The training data is illustrated by colored dots that represent the five classes

Another 40 data points were selected from each class to form the test data with 200 sample points in total. A Residual Neural Network was trained. The width of each layer nc was set to 8 by duplicating the original two features (x_1 and x_2) four times; the number of time steps of the network nt and the total time T were subject to change. Furthermore, the tanh activation function and the softmax hypothesis function were applied.

ResNet with different time steps

The ResNet structure we applied is modified based on the standard ResNet by the authors of [5]. Different from the original ResNet, two terminologies are introduced, which are the control layer nt, and the state layer nY. The control layers are the discretized time points for the weights, and the state layers are those of the underlying ODE with the continuous interpretation. The two time points can be discretized separately with respect to the total time T. Figure 3.2 illustrates the main idea. In particular, while we fix the step size between two control layers, we change the number of state layers over a fixed amount of time to see the difference in the classification performance of each network training.

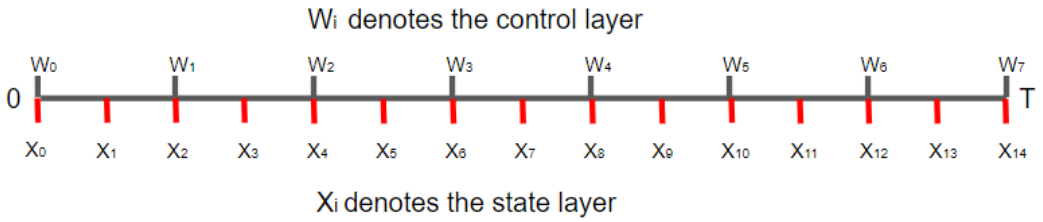

Figure 3.2: The modified ResNet can separately discretize the weights and its underlying ODE. The number of control layers indicates the number of time points we use to discretize the weights that needed to be trained. The number of state layers indicates the number of time points we use to discretize the ODE.

3.1 Classification Performance of ResNet with Different Numbers of State Layers

The purpose of conducting the first experiment is to illustrate the continuous interpretation of DNNs proposed in [5] on a concrete example. The experiment was conducted on the Peaks example [5]. A modified ResNet introduced in Chapter 3 was trained. With the modified ResNet, we are able to see how different number of discretized time points of the underlying ODE are going to affect the classification performance of a network.

The total time T is set to 20, the width of each layer nc is set to 8, and the number of control layers nt is set to 16. The same group of data samples and initialized weights and biases are used for every training. We first train a network with 8 state layers. After we finish training the network, we record the trained weights along with the loss function, which if combined together will generate the classification result. Then, we keep everything fixed but double the number of state layers, and we again train the network and record the trained weights and the loss function. We repeat this procedure six times until a network with 256 state layers is trained and the results are recorded. Coding details of the network setup and how the weights and loss functions are documented for each training are listed in Chapter 5.

After we trained the network, six sets of solutions (trained weights) and the corresponding loss functions were recorded. The classification result is one way to examine the performance of a neural network. Thus, we show

the classification results of the combination of the six loss functions and six solutions. A total number of thirty six classification results are displayed in Figure 3.3. Coding details that illustrate how the results are computed and plotted are referenced in Chapter 5.

Classification results for different number of time steps

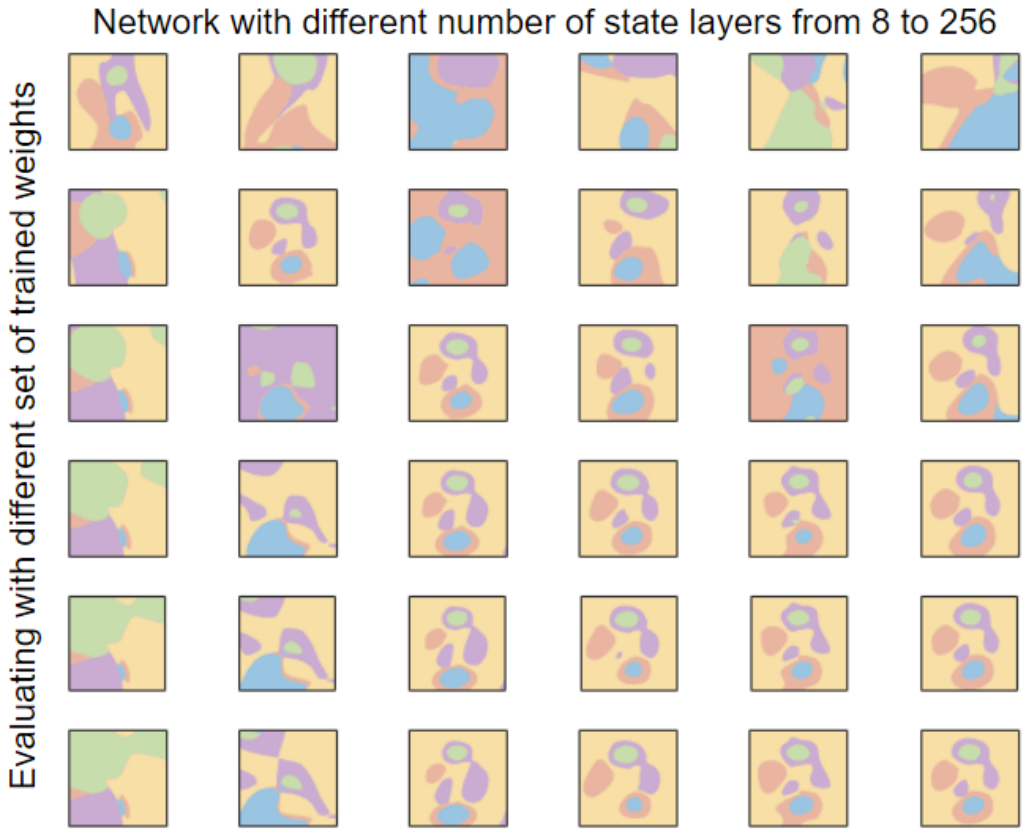

Figure 3.3: Classification results of each loss function combined with each set of weights. The first column shows the classification results generated from the loss function of the network with 8 state layers fit with all six sets of weights, while the last column shows that of the network with 256 state layers. The diagonal indicates the original prediction made from each time of training.

Result

From the figure, we can observe that the step size of the discretized time points of the underlying ODE, i.e., the number of state layers, does affect

the network's classification performance. To be specific, if we take a closer look at the first column, we see that even though the original classification results show some level of accuracy, the network fails to classify when the trained weights from other trainings are applied. Furthermore, when the trained weights of the network with 8 state layers are fit into other loss functions, the predictions are not satisfying. The same pattern is observed for the second column and the second row, which both correspond to the 16-layer network. On the other hand, the classification results in the bottom right 3×3 subplots are much better than the results on the left. Moreover, we can observe less difference between the results as the trained weights and the loss functions are combined together. From the observations, we conclude that the number of state layers in a network affects the accuracy of a network's prediction on the data, and the more state layers a network has, the better classification results a network will generate. In addition, we state that in order to ensure the accuracy of the classification result, a network must have a certain number of state layers subject to the total time T.

3.2 Stability of the ResNet Forward Propagation

The purpose of conducting the second experiment is to study the stability of neural network forward propagation described in (2.16) by visualizing the eigenvalues of the network on the Peaks example. Furthermore, we would like to explore whether there is a relationship between the depth of a neural network and the stability of the forward propagation. In the experiment, we first achieved the Jacobian matrices of each layer of the network and then computed the corresponding eigenvalues and visualized them. The experiment was run with the Peaks data set described in Figure 3.1. A standard ResNet was trained with a total time $T = 20$ and a layer width nc $= 8$ (duplicate the original two features by four times). The Jacobian matrix of each layer was computed based on the code listed in Chapter 5.

After we computed the Jacobian matrix from each layer, we applied the $eig()$ function in MATLAB® to find the eigenvalues of each matrix. Then, we plotted the eigenvalues. As mentioned in Section 2.4, a stable discretization method implies the good approximation of the discrete problem to the continuous ODE. Furthermore, as the authors of [5] observed,forward Euler is stable only under the condition that

$$|1 + h \max \lambda(J_i)| < 1 \text{ for } i = 0, 1, \ldots, N - 1 \tag{3.1}$$

In order to evaluate the eigenvalues more straightforwardly, we constructed the unit ball centered at (-1,0) in the complex z-plane with real numbers on the x-axis and imaginary numbers on the y-axis along with the eigenvalues while we do the plotting. The formula to compute the Jacobian matrices is mentioned in (2.15) and shown in code in Chapter 5. Six networks were trained, where each network was discretized with a different number of time points. After the previous network has been trained, the number of time points was doubled to discretize next network. The weights of each layer were stored and used for computing the eigenvalues. Three networks were trained in this experiment, the total time T is set to 10 and the layer width nc is set to 8. The first network is discretized with 8 time points, while we halve the step size, i.e., discretize the network with 16 time points for the second network, and again half the step size for the third one. Figure 3.4 shows the distribution of the eigenvalues of the network with 8 control layers. The red circle indicates the stability region. Figure 3.5 shows the eigenvalues of the 16 layer network. Figure 3.6 shows 16 of the 32 sets of eigenvalues for illustration purposes. The written codes for plotting the eigenvalues and the stability region of forward Euler discretization method are listed in Chapter 5.

Visualization of the Eigenvalues of networks with different time step sizes

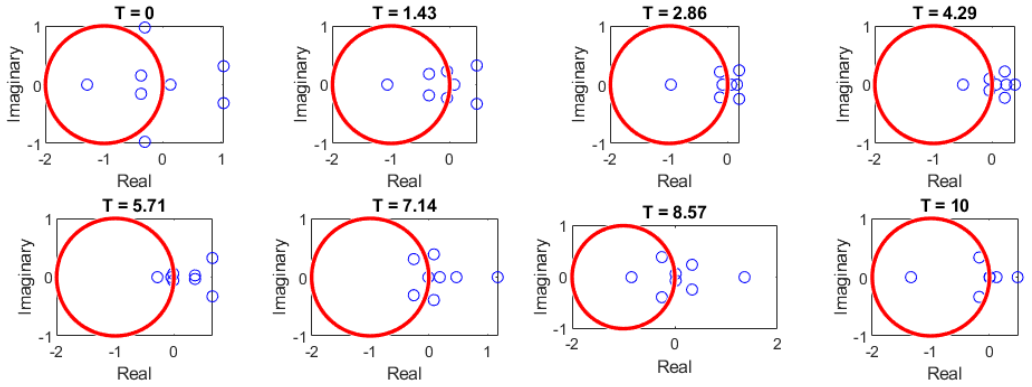

Figure 3.4: The subplots above are the distribution of the eigenvalues of the trained weights of each control layer of the first network, which is discretized with 8 time points.

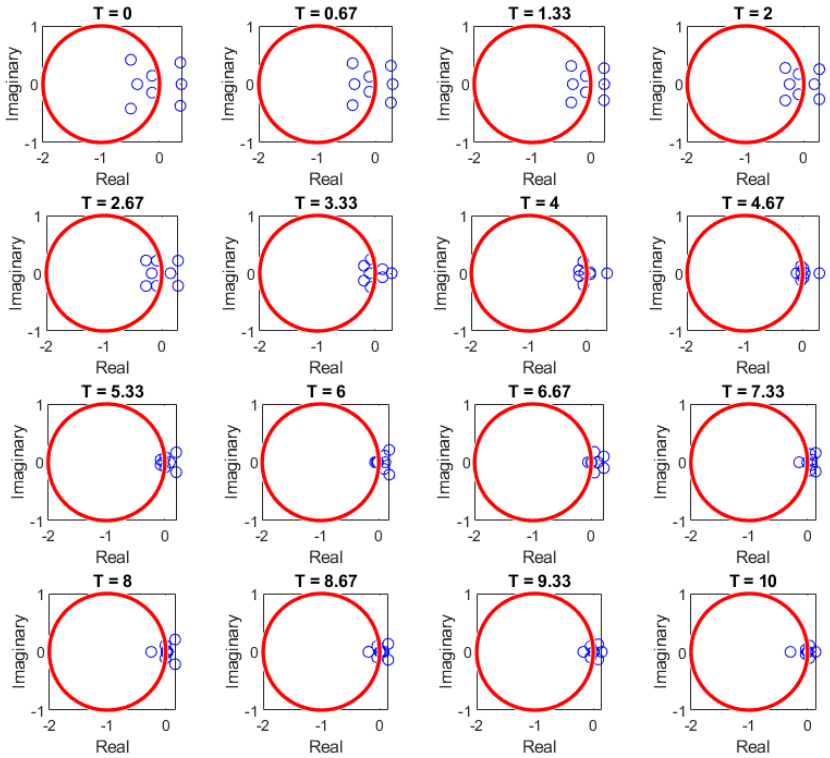

Figure 3.5: The subplots above are the distribution of the eigenvalues of the trained weights of each control layer of the second network, which is discretized with 16 time points.

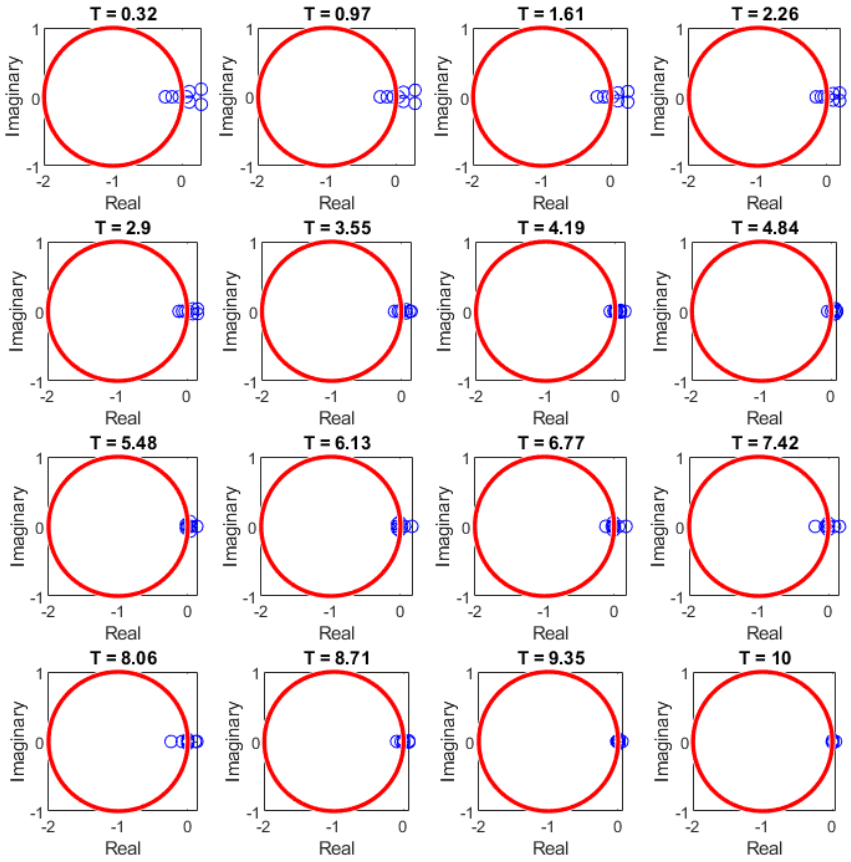

Figure 3.6: The subplots above are the distribution of the eigenvalues of the trained weights of selected 16 control layers of the third network, which is discretized with 32 time points.

Result

From the distributions, we can observe that the eigenvalues of a network discretized with fewer time points are more spread out, and there exist eigenvalues with a real part much larger than zero. This observation indicates

the instability of forward propagation. As we increase the number of time points, we detect less violation of the stability condition as shown in Figure 3.5 as more eigenvalues than that of the first network fall into the stability region and fewer eigenvalue with positive real parts are observed. In Figure 3.6, we can see that the eigenvalues meet the stability condition stated in 2.16. Based on the observations, we conclude that there exists a relationship between the step size and the stability of the network's forward propagation. With a fixed total time T, a decrease in the step size, will lead to an increase in the stability of the network's forward propagation.

3.3 Visualizing the Loss Function

As well-explained in [10], the training of neural networks relies heavily on the minimizers found by solving the loss function introduced in section 2.2.1. However, a lack of visualization of the loss function has constrained the studies to remain at an abstract and theoretical level. In the third experiment, we first aimed to implement the visualization method introduced in [10] with minor modifications to visualize the curvature of the loss landscapes of the trained neural networks on the Peaks example. We also want to determine how different network architectures affect the difficulty of minimizing a loss function.

In [10], the author first showed us the limitation of 1-Dimensional Linear Interpolation, which is the difficulty to visualize non-convexities, and then suggested using 2D Contour Plots for better understanding of the loss landscape. In order to realize the 2D plot, we first need to choose one center point θ^* and two direction vectors ξ and ζ. Then we plot the function

$$f(\alpha, \beta) = L(\theta^* + \alpha\xi + \beta\zeta) \tag{3.2}$$

using researcher's choice of scalar parameters α and β.

The two direction vectors ξ and ζ, are both $\in \mathbb{R}^n$. Here, n is the total number of paramatrized weights in the network. Each entry of the two vectors is sampled from a random Gaussian distribution in the range of $[0, 1]$. Though it seems easy to plot the loss landscape with the "random directions," the approach would not be able to be used for comparing two landscapes before the directions are normalized using the filter normalization method [10]. The reason we must normalize the directions is that the network weights are invariant to scaling due to the use of batch normalization, i.e., a network's behavior would remain unchanged if we just re-scale the weights. The filter-normalization method is introduced as

$$d_{i,j} \leftarrow \frac{d_{i,j}}{||d_{i,j}||}||w_{i,j}||, \tag{3.3}$$

where $d_{i,j}$ represents the j^{th}filter of the i^{th} layer of d, and the $||\cdot||$ is the Frobenious norm. In our experiment, we normalized the vectors and weights

with their averages instead of applying the Frobenious norm for simplicity. The code for plotting the loss landscape is listed in Chapter 5. The total time T was set to 5 and the width of each layer nc was set to 8. We plotted the loss landscapes of trained networks with different depth. Figure 3.7 is the loss function of a standard 8-layer ResNet. Figure 3.8 below is the loss function of a standard 20-layer ResNet with the same total time and layer width.

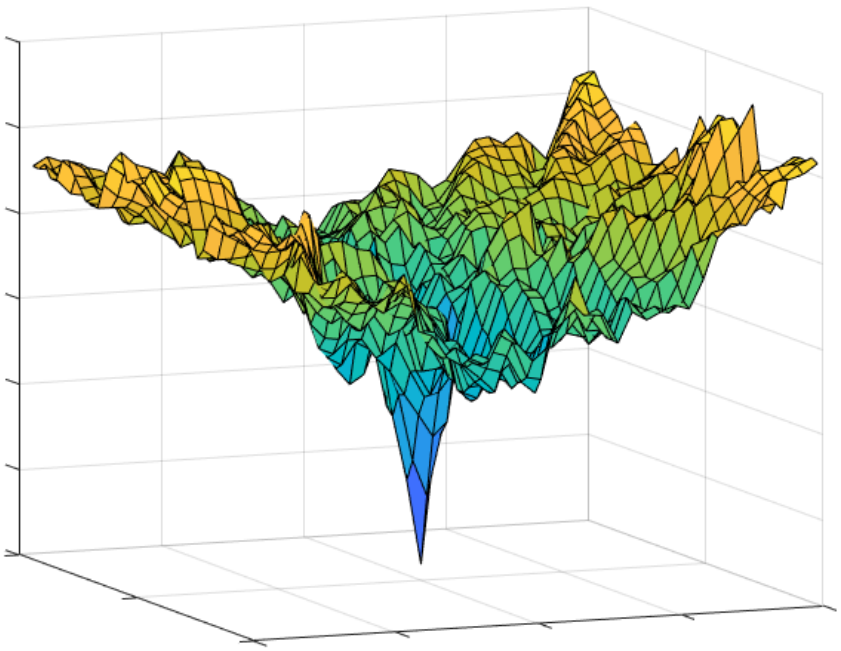

Figure 3.7: The loss function of an 8-layer standard ResNet. Lots of non-convexities and a lack of smoothness can be observed.

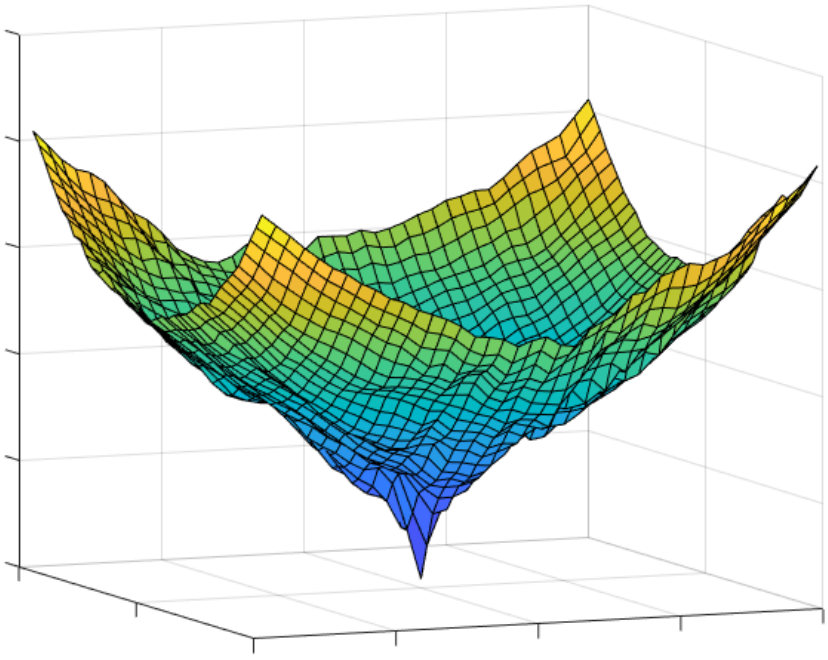

Figure 3.8: The loss function of a 20-layer standard ResNet. Much more smoothness can be observed from this landscape than from that of the 8-layer network.

Result

With the visualization method from [10], we are now able to study the properties of networks' loss functions more directly. We observe more non-convexities and less smoothness from the shallower network than from the deeper one. If a loss function is more smooth and more convex, it is easier to find the global minimizer, i.e., the network can be trained more efficiently. This observation indicates that the increase of the number of layers in a network would decrease the difficulty of finding the global minimizer and in-

crease the chance of achieving a good generalization from the loss function. This finding suggests that a ResNet with deeper structure could be trained more efficiently.

3.4 RK4 Forward Propagation

There are several advantages that we can obtain from the ODE interpretation of DNNs proposed in [5]. Most importantly, now we have a general guideline for interpreting and constructing the network framework with the abundant knowledge achieved from a more than 120 years of development of accurate and efficient ODE solvers. With this novel approach, many works have been accomplished by various researchers such as [3, 11], but the field of higher order numerical methods, such as higher order Runge Kutta methods, are rarely explored. Theoretically speaking, a higher order numerical method could solve the problem with a lower truncation error [1]. Since the decrease of a truncation error would lead to an increase of the accuracy, it is worth-trying to design a new neural network structure based on higher order forward propagation algorithms.

In our experiment, we modified the forward propagation of a standard ResNet with RK4 method. The coding details can be found in Chapter 5. The first goal of our experiment is to ensure the correctness of our new algorithm by comparing the objective functions of the two forward propagations. The package used is an example from Dr. Lars Ruthotto's numerical deep

learning course. The input data and class label were generated before the network was trained and were stored for future runs in order to maintain consistency. Then, the two propagation algorithms were trained at the same time with different time steps. The effect of the total time and the time step size of each propagation was observed from the objective functions respectively. The total time T was set to 10. In order to check the correctness of our algorithm, we set the time step h of the RK4 propagation to be four times bigger than that of the ResNet. The reason behind this increased time step assignment is because since each step in a RK4 method is evaluated four times, then the objective function of a RK4 forward propagation network with four-time bigger time step should be identical to the one of a standard ResNet. Figure 3.9 illustrates the results.

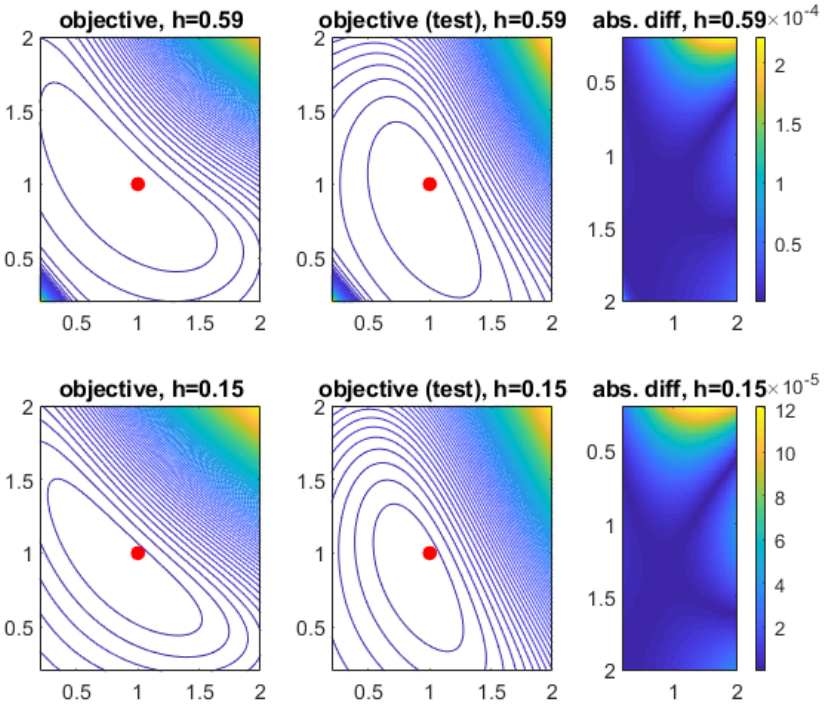

Figure 3.9: The first column includes the objective functions generated from the training data, and the second column includes the objective functions achieved from the test data. The first row indicates the objective functions of a RK4 implemented forward propagation network, and the second row shows the ones of a standard ResNet.

Result

A high level of similarity can be observed from the objective functions. The causes behind the slight differences between graphs are yet to be determined. However, due to the time limitation, we did not have a chance for more in-depth experiments to further validate the algorithm and apply the algorithm on other examples.

Chapter 4

Summary and Conclusion

In this thesis project, we look further into the continuous interpretation of Deep Neural Networks (DNNs) and study on the stability of a DNN and how does the network structure affect network training and performance.

In the first experiment, we applied the continuous interpretation of DNNs on the Peaks example. Different from a standard ResNet, we discretized the network and the weights separately, which gave us the state layer with respect to the network and the control layer with respect to the weights. We trained six networks with different number of state layers, and recorded the weights and the loss function of each training. Then, we combined the six set of weights and the six loss functions together, where each pair of weights and loss function generated one classification result. Looking at all thirty six classification results, we concluded that the increase of a network's state layers will increase the network's classification performance.

In the second experiment, we explored the connection between network structure and the stability of the forward propagation by looking into the stability of the network's underlying ODE. Six networks with different number of layers were trained. For each network, we computed the eigenvalues of the Jacobian matrix of each layer based on the stability theories of ODEs. Then, the eigenvalues were plotted with the stability region. From the results, We observed that the increase of number of layers of a network will lead to the increase of the stability of the forward propagation, as the eigenvalues of the network are more convergent to the stability region with the increase of the network depth. This observation leads us to the conclusion that the stability of a ResNet will increase if the number of layers in the network increases.

In the third experiment, a technique of visualizing a neural network's objective function was implemented. With this visualization method, the relationship between network structure and the training efficiency can be studied more directly. we observed that as the number of layers of a network increases, more smoothness and convexities can be observed from the objective function, i.e., the network can be trained more efficiently.

In the last experiment, we developed a new forward propagation algorithm based on the Runge Kutta method order of four (RK4). Since RK4 is a higher order numerical method than forward Euler, a neural network with RK4 forward propagation should perform with higher accuracy than a standard ResNet can do. Due to the time limitation, we only had chance to test the correctness of our algorithm by comparing the objective functions of

a RK4 network and a ResNet, while the step size of the RK4 network is four times bigger than that of the ResNet. The result has proven the correctness of our algorithm, but more in-depth experiments are needed to prove that the RK4 forward propagation works superior than the original algorithm.

This thesis project mostly aims at finding the relationship of a network's structure and its training. Some progress has been made throughout the project, and our results will surely motivate future studies of DNNs from a mathematical angle.

Chapter 5

MATLAB® Code

Network setup

```
%% setup network
T  = 20;   % final time
nt = 8;  % number of time steps (number of theta in aResNN)
nc = 8;   % number of channels (width)
h = T./(nt-1); %size of each time step
nYs = 2.^(3:8); %choices for the number of layers
LossF = cell(numel(nYs),1);
%the loss function of each network is recorded
for k = 1 : numel(nYs)
nY = nYs(k); %number of layer we use in the experiment
hY = T./(nY-1); % time step of nY
```

Network training and data documentation

This piece of code is selected from the MegaNet package.The code explains how the network is trained and how the weights and loss functions are documented.

```
%% setup objective function with training and validation data
fctn = dnnVarProObjFctn(net,pRegTh,pLoss,pRegW,
            classSolver,Ytrain,Ctrain);
fval = dnnObjFctn(net,[],pLoss,[],Yv,Cv);
floss = dnnObjFctn(net,[],pLoss,[],Ytrain,Ctrain);
LossF{k} = floss;


%% solve the problem
if k == 1
    th0       = 1e-1*initTheta(net);
end
diary([mfilename '-nY-' num2str(nY) '.txt']);
diary on;
[thetaOpt,His]  = solve(opt,fctn,th0,fval);
thetaOpt  = solve(opt,fctn,th0,fval);
[Jc,para] = eval(fctn,thetaOpt);
WOpt      = reshape(para.W,[],5);
n = 16;
```

```
thetaOpen   = thetaOpt(1:16); %Weights of the open layer
thetaResNN  = reshape(thetaOpt(17:end),[nc.*nc+1,nt]);
%Weights of the ResNet
thetaResNNK = reshape(thetaResNN(1:64,:),nc,nc,nt);
thetaBout = thetaResNN(65,:);
Weight = cell(nt,1); % get the weight matrices
for i = 1 : nt %total number of weight matrices
    Weight{i} = thetaResNNK(:,:,i);
end
save([mfilename '-nY-' num2str(nY)],'thetaOpt','His','WOpt');
diary off;
```

Plot results

The following piece of code is selected from the MegaNet package.The code chunk shows how the classification results are computed and plotted.

```
%% plot results
%load('Run4.mat'); %get the solution of each case
k = 0;
for i = 1 : 6
```

```
    for j = 1: 6
        [Ydata,Yn,tmp] = apply(LossF{i}.net,
                Run4{j}.thetaOpt,Yv);
        k = k+1;
        subplot(6,6,k);
        viewContour2D([-3 3 -3 3],
                Run4{j}.thetaOpt,Run4{j}.WOpt,
                LossF{i}.net,LossF{i}.pLoss);
        axis equal
        hold on
        viewFeatures2D(Yv,Cv);
        title(['nY-',num2str(2.^(2+j)),'Loss-',
                num2str(2.^(2+i))]);
    end
end
```

Getting the weights

This piece of code is select from the MegaNet package.The code chunk computes the Jacobian matrix of each layer.

```
thetaOpen   = thetaOpt(1:16); %Weights of the open layer
thetaResNN  = reshape(thetaOpt(17:end),[nc.*nc+1,nt]);
%Weights of the ResNet
```

```
thetaResNNK = reshape(thetaResNN(1:64,:),nc,nc,nt);
thetaBout = thetaResNN(65,:);
Weight = cell(nt,1); % get the weight matrices
for i = 1 : nt %total number of weight matrices
    Weight{i} = thetaResNNK(:,:,i);
end
```

section*Computing the Jacobian matrix This piece of code is selected from the MegaNet package.The code chunk that computes the Jacobian matrix of each layer.

```
%% compute the Jacobian of each layer
Main idea is to get the derivative of \sigma(K(t).'*y(t))
[Ytdata,~,~] = apply(net,thetaOpt,Ytrain(:,2));
Jacobian = cell(nt,1);
%Cell array to store Jacobian matrix for each layer
Y_Data = cell(nt,1); %Cell array to store Ys' for each layer
Yj = Ytrain(:,2);%initial value of Y
thetaOpen = reshape(thetaOpen,[8,2]);
[Yj,~] = tanhActivation(thetaOpen*Yj); %First block to open up
 for i = 1 : nt
    Yj = Yj + h.*(tanhActivation(Weight{i}*Yj)+
         ones(nc,1)*thetaBout(1,i));%input for next layer
    Y_Data{i} = Yj;
    [~,dYj] = tanhActivation(Weight{i}*Yj);
```

```
    Jacobian{i} = diag(dYj)*Weight{i};
end
```

Check stability

```
for k = 1:nt
    A = Jacobian{k};
    e = h.*eig(A);
    centers = [-1,0];
    radii = 1;
    figure(1);
    subplot(4,4,k);
    plot(real(e),imag(e),'bo')
    % Plot real and imaginary parts
    viscircles(centers,radii);
    daspect([1 1 1])
    xlabel('Real')
    ylabel('Imaginary')
    t1 = ['layer' num2str(n)];
    title(t1)
end
```

Visualizing Loss Function

```
n = 2.*nc+nt.*(nc.*nc+1);

d1 = normrnd(0,1,[n,1]);

d2 = normrnd(0,1,[n,1]);

Q = zeros(9,n);

Q(1,1:16) = 1;

dim = nt+1;

for i = 2 : (dim)

    Q(i,17+(65*(i-2)):17+65*(i-1)-1) = 1;

end

total = zeros(dim,1);

for i = 1:dim

    total(i,1) = sum(Q(i,:));

end

d1Average = (Q')*(Q*d1)./(Q'*total);

d2Average = (Q')*(Q*d2)./(Q'*total);

thetaAverage = (Q')*(Q*thetaOpt)./(Q'*total);

d1normed = (d1.*thetaAverage)./d1Average;

d2normed = (d2.*thetaAverage)./d2Average;

Viewer = zeros(20,20);

for a = 1:40

    for b = 1:40
```

```
        thetaNormed = thetaOpt + ((.05*a-1)*d1normed)
        + ((.05*b-1)*d2normed);
        Viewer(a,b) = eval(fctn,thetaNormed);
    end
end
```

Standard ResNet Forward Propagation

```
th   = linspace(0.2,2,nth);
Phic  = zeros(nth,nth);
Phict = zeros(nth,nth);
for k1=1:nth
    k1
    for k2=1:nth
        Cpred = Y0;
        Cpredt = Y0t;
        K = getK([th(k1);th(k2)]);
        for j=1:N(1)
            Cpred  = Cpred + h(1)*activation(K*Cpred);
            Cpredt = Cpredt + h(1)*activation(K*Cpredt);
        end
        Phic(k1,k2)  = 0.5*norm(Cpred-C,'fro')^2;
        Phict(k1,k2) = 0.5*norm(Cpredt-Ct,'fro')^2;
```

```
    end
end
```

RK4 Forward Propagation

```
th   = linspace(0.2,2,nth);
Phic  = zeros(nth,nth);
Phict = zeros(nth,nth);
for k1=1:nth
    k1
    for k2=1:nth

        Cpred = Y0;
        Cpredt = Y0t;
        K = getK([th(k1);th(k2)]);
        for j=1:N(1)
            R1 = activation(K*Cpred);
            R2 = (h(1)/2)*activation(K*(Cpred + (h(1)/2)*R1));
            R3 = (h(1)/2)*activation(K*(Cpred + (h(1)/2)*R2));
            R4 =  h(1)*activation(K*(Cpred + h(1) * R3));
            Cpred = Cpred + (h(1)/6)*(R1+2*R2+2*R3+R4);

            Rt1 = activation(K*Cpredt);
```

```
            Rt2 = (h(1)/2)*activation(K*(Cpredt + (h(1)/2)*Rt1));
            Rt3 = (h(1)/2)*activation(K*(Cpredt + (h(1)/2)*Rt2));
            Rt4 =  h(1)*activation(K*(Cpredt + h(1) * Rt3));
            Cpredt = Cpredt + (h(1)/6)*(Rt1+2*Rt2+2*Rt3+Rt4);
        end
        Phic(k1,k2)  = 0.5*norm(Cpred-C,'fro')^2;
        Phict(k1,k2) = 0.5*norm(Cpredt-Ct,'fro')^2;
    end
end
```

Bibliography

[1] U. M. Ascher and C. Greif. A First Course on Numerical Methods. SIAM, Philadelphia, 2011. 3, 5, 15, 33

[2] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018. 5, 6, 9

[3] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6572–6583, 2018. 33

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 5

[5] E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017. 3, 5, 6, 9, 11, 13, 14, 15, 17, 18, 20, 21, 25, 33

[6] E. Haber, L. Ruthotto, E. Holtham, and S. Jun. Learning across scales - multiscale methods for convolution neural networks. In *Proceedings of*

the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 3142–3148, 2018. 5, 12, 15

[7] 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*, pages 770–778, 2016. 2, 5, 10

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. 2

[9] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 10

[10] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6391–6401, 2018. 5, 11, 17, 29, 30, 32

[11] Y. Lu, A. Zhong, Q. Li, and B. Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 3282–3291. JMLR.org, 2018. 33

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2

[13] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR*, 2014. 2