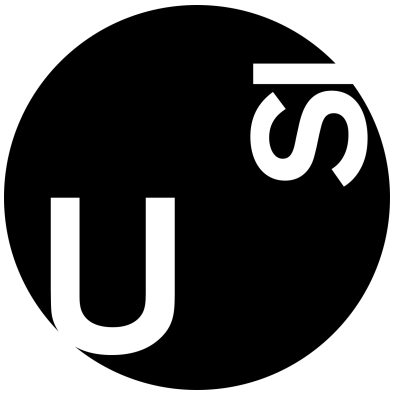

Training of ResNets using Multi-level Methods and Ordinary Differential Equations

Master Thesis

Qianbo Zang

28 August, 2023

Advisors: Prof. Dr. Rolf Krause

Co-Advisors: Dr. Hardik Kothari, Dr. Giacomo Rosilho de Souza

Faculty of Informatics, USI Lugano

Abstract

In this thesis, we propose to train Residual Networks (ResNets) using two different methods: multi-level and numerical methods for ordinary differential equations (ODEs). Inspired by the dynamical system point of view of ResNets, we can leverage multi-level and numerical methods for ODEs to train ResNets. Our experimental results show that the multi-level methods can effectively reduce the training time while ensuring good model accuracies. More accurate numerical methods for ODEs can improve the model accuracies but only for regression problems.

Contents

Chapter 1

Introduction

Deep learning has revolutionized many fields, from anomaly detection to computer vision. Among the various architectures that have emerged in this domain, Residual Networks (ResNets) stand out due to their capacity to train intense networks without the hindrance of vanishing or exploding gradients. However, as with many deep learning models, the training of ResNets can be computationally intensive and time-consuming, significantly as the depth of the network increases.

The dynamical system view of ResNets offers a unique lens through which we can view and understand ResNet's behaviors. This viewpoint corresponds the layers of a ResNet to a discrete dynamical system, evolving over time. Such a perspective naturally leads to two questions: Can techniques from dynamical systems, particularly the multi-level methods, be used to accelerate the training process of ResNets? Can numerical methods for ordinary differential equations (ODEs) be leveraged to enhance the accuracy of ResNets?

In this thesis, we will explore these two questions. In the beginning, drawing inspiration from the multi-grid methods, we propose a novel approach to train ResNets using multi-level methods. These methods have the potential to offer significant computational advantages when applied to the training of deep neural networks.

Furthermore, we delve into the accuracy of various numerical methods for ODEs and their implications on model performance. While it is intuitive to believe that more accurate numerical methods would invariably lead to better model accuracies, our findings suggest that this is the case predominantly for regression problems.

Our primary objective is to discover a method that not only accelerates the training process of ResNets but also ensures that there is no compromise on model accuracy. Through extensive experimentation, we aim to provide a robust solution that matches the efficiency of dynamic system techniques

with the precision required in deep learning, setting a new benchmark for ResNet training methodologies.

Chapter 2

Artificial Neural Networks

This chapter aims to provide an overview of Artificial Neural Networks, specifically Multiple-Layer Perceptrons, Convolutional Neural Networks, and Residual Neural Networks. These networks are the most critical Artificial Neural Networks in research and industry, especially in Computer vision. Additionally, this chapter covers the Universal Approximation Theorem, which forms the mathematical foundation of Artificial Neural Networks.

2.1 Multiple-Layer Perceptrons (MLPs)

Computer scientists and neuroscientists built perceptron models based on biological neurons. In terms of structure, the perceptron imitates biological neurons. For example, there are multiple inputs and one output. The output results will be used as input for other units. A perceptron is a simple mathematical model that does number crunching [1]. In this chapter, we will introduce the structure of Multiple-Layer Perceptrons and loss functions with gradient descent optimization methods.

2.1.1 Structure of One Perceptron and MLPs

Mathematical Definition of Perceptrons

One perceptron only has a specific fitting ability and can perform binary classification on the inputs. Mathematically, given an input column vector:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix},$$

an weight column vector:

$$\mathbf{w} = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{pmatrix},$$

and a bias b, one perceptron can be define as a operation performed by weighted sum $z = \mathbf{w}^T\mathbf{x} + b$ and activation processing, given as:

$$\sigma(\mathbf{w}^T\mathbf{x} + b).$$

Activations

Activation is used to add non-linearity into the output of a neuron. This non-linearity help neural networks to simulate the patterns of complex, non-linear relationships in datasets. Without an activation function, a neural network would be a linear regression, limiting its capacity only to model linear relationships. Because according to the matrix decomposition and linear combination, a sequence of linear models can be merged into one linear model that is just a linear regression.

Here are some of the most commonly used activation functions:

- Sigmoid Function:
$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$
- Rectified Linear Unit (ReLU) Function:
$$\text{ReLU}(z) = \max(0,\ z).$$
- Softmax Function:
$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

Bottleneck of One Perceptron

Binary classification means to divide the input data into two categories. Boolean operations are logical operations and can be viewed as binary classification operations. There are three weight parameters, namely ω_1, ω_2, and bias b. We can adjust the values of these three parameters so that one perceptron can simulate Boolean Operations. For example, as Figure 2.1, consider the input of one artificial neural is x_1 and x_2, and four different input combinations are shown in the table below. We can set $w1 = 0.5,\ w2 = 0.5,\ b = -0.8$ to

$$w_1x_1 + w_2x_2 + b = (w_1\ w_2)\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + b.$$

In this case, we can simulate the ”and” Boolean Operation.

x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

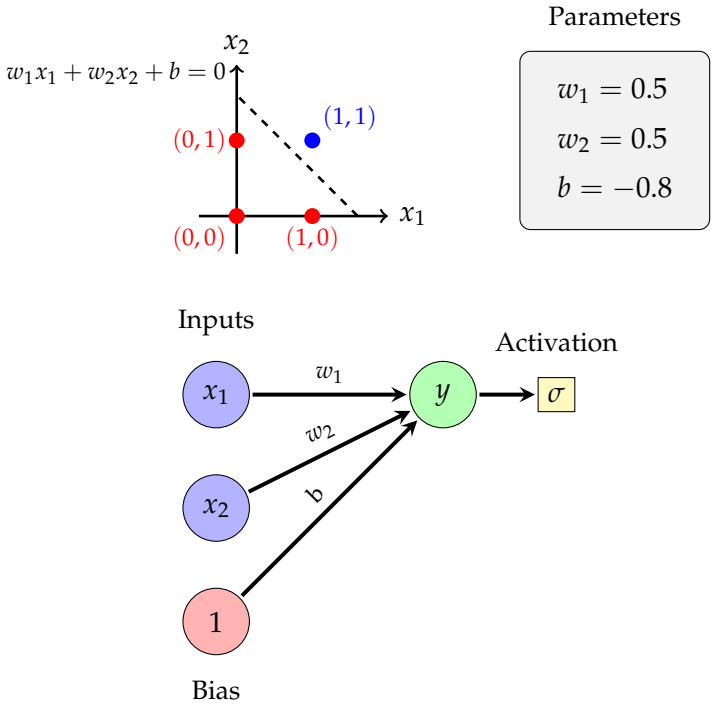

Figure 2.1: A visualization of solving "and" boolean operations by one perceptron. x_1 and x_2 are the inputs of this artificial neural. $x_1,\ x_2 \in \{0,\ 1\}$. The output of this artificial neural is "$w_1x_1 + w_2x_2 + b$", which is also the answer of "x_1 and x_2".

However, there is no way for a perceptron to imitate complex logical operations, such as ”exclusive or.” However, multiple perceptrons can be linked together, enabling them to mimic complex logic operations called Multiple-Layer Perceptrons.

Multiple-Layer Perceptrons

A single perceptron processes its input to produce a single output. In comparison, multiple perceptrons processing their inputs in parallel form a layer of MLP and produce multiple outputs. Mathematically, for a single layer MLP with m perceptrons, the output of the layer can be represented as a vector $\mathbf{y}$:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix},$$

where each element corresponds to the output of a single perceptron $y_i = \sigma(\mathbf{w}_i^T \mathbf{x} + b_i)$.

Generally, the first layer, middle layers and last layer are named input layer, hidden layers and output layer respectively. There is no specification for MLP about the number of hidden layers, so the appropriate number of hidden layers can be selected according to actual processing requirements. And there is no limit to the number of neurons in each layer in the hidden layers or output layer.

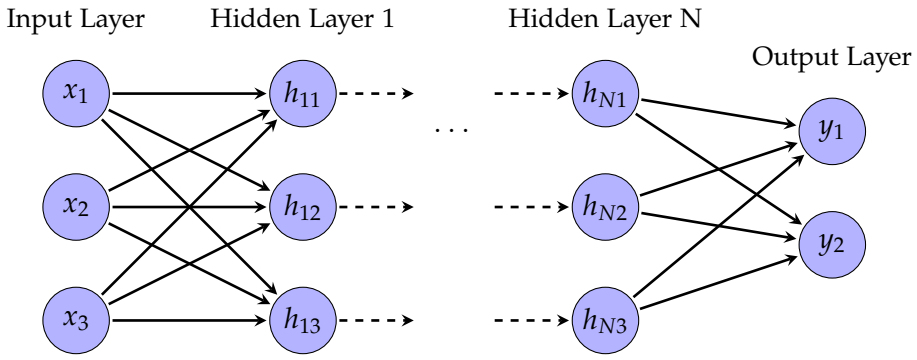

Figure 2.2: Structure of a MLP. It includes three parts: the input layer, hidden layers, and output layer. The input layer and output layer are unique, but the number of hidden layers is not restricted.

2.1.2 Loss Functions and Model Training

Loss Functions

In deep learning, loss functions are one of the most important methods to measure the performance of a model. Different models generally use different loss functions. The two most common loss functions are the mean-squared error (MSE) loss function and the cross-entropy (CE) function.

The MSE function is used to measure the distance between sample points $(y_1, y_2, \ldots, y_n)$ and the predicted value $(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_3)$, given as:

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

The MSE has become an excellent distance measure due to the advantages of no adjustable parameters, low computational cost, and precise physical meaning. Although MSE performs weakly in image and speech processing, it is still a standard for evaluating signal quality. The MSE function is mainly used in regression problems.

The CE function is a concept in information theory initially used to estimate the average coding length. It is now used to evaluate how the probability distribution obtained from the sample points $(y_1, y_2, \ldots, y_n)$ differs from the predicted value $(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_3)$, given as:

$$\text{CE} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(softmax(\hat{y}_i)).$$

The CE loss function is the most frequently used loss function for classification in convolutional neural networks.

Model Training

The training of a deep learning model refers to finding a set of model parameters so that the predicted values obtained from the data input are as close as the actual values obtained from sampling. The loss function is used mainly in the training stages. By minimizing the loss function, we transform the model's training into an optimization problem, given as:

$$\min_{\omega} \mathcal{L}(\omega),$$

where min denotes the minimization, ω denotes the parameters of the model, and $\mathcal{L}$ denotes the loss function.

2.1.3 Gradients Descents (GD) and Back Propagation (BP)

Gradients Descents

In deep learning, the optimization problems transformed from model training are computed by GD methods. Gradient descent is used to adjust the model parameters iteratively, thereby minimizing the loss function $\min_{\omega} \mathcal{L}(\omega)$. In an iteration, the gradient descent method first calculates the gradient $\nabla\mathcal{L}(\omega_{\text{old}})$

of the loss function at the current parameter position and then moves the parameter a small step in the negative direction of the gradient. The learning rate parameter usually determines this step size, , given as:

$$\omega_{\text{new}} = \omega_{\text{old}} - \alpha \nabla \mathcal{L}(\omega_{\text{old}}),$$

where ω_{new} denotes the new parameter position of the deep learning model, ω_{old} denotes the old parameter position, and α denotes the learning rate. Then, we compute the new gradient at the new parameter position and repeat the process until a satisfactory solution reached or a specific stopping condition happens. For example, the gradient has fully converged and cannot continue to descend.

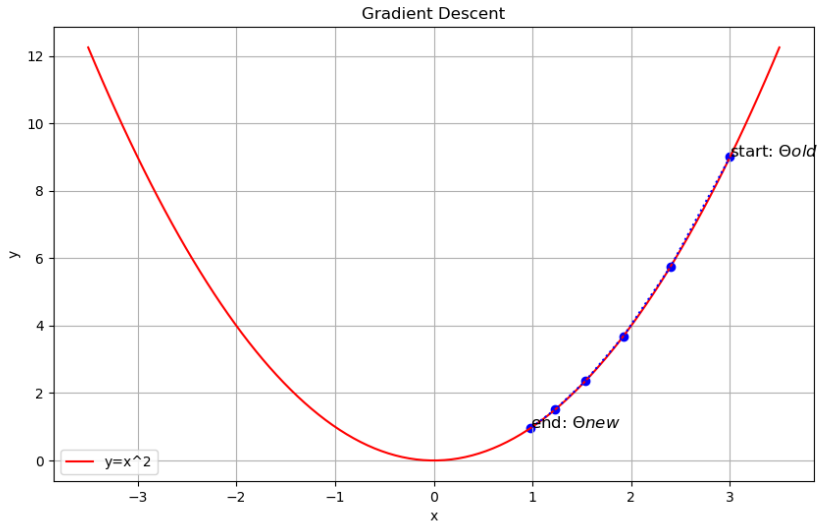

Figure 2.3: An example of Gradients Descent to find the minimum point of $y = x^2$. We start from a random point and then move the parameter a small step in the negative direction of the gradient. Until five epochs, we get a new point that is closer to the minimum point.

However, the gradient descent method is not guaranteed to find a globally optimal solution. Especially when we deal with non-convex optimization problems, in this case, the gradient descent can only find a locally optimal solution probably, and this locally optimal solution may be some distance away from the global optimum.

Back Propagation

From Chapter 2.1.2, we learn that the training process of a deep learning model is to solve the optimization problem. According to the previous section,

we use gradient descent to solve this optimization problem. Therefore the difficulty of model training is transformed into a gradient computation problem. In gradient computation, we use the chain rule when the function is too complex. Deep Neural Networks fall into the category of very complex functions. Backpropagation is proposed and used for gradient computation based on the chain rule for efficiently computing the gradient of the loss function concerning all weights and deviations in the network [2]. BP consists of two main stages: forward propagation and backpropagation. Forward propagation means the input data is passed from the deep learning model to get the predicted values. BP computes the derivative of the error concerning the weight of the last layer. Then, it computes the derivative with respect to the previous layer and so on, using the chain rule.

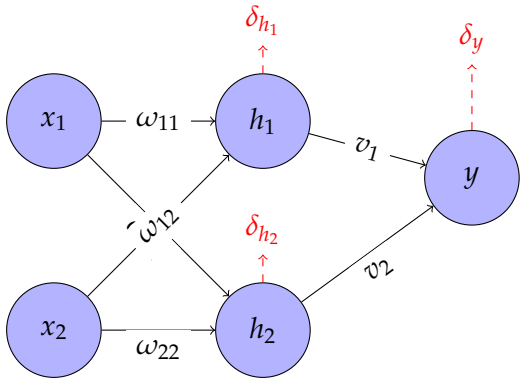

Figure 2.4: An example of BP. Firstly, calculate the error in the output layer $\frac{\partial L}{\partial y}$. Then, propagate this error backward to compute $\frac{\partial L}{\partial h_i}$, and $\frac{\partial L}{\partial x_i}$.

According to Figure 2.4, for example, we use Mean Squared Error (MSE) $L = \frac{1}{2}(y-t)^2$ as the loss function. Here, we will show the basic process of BP. For each neuron, there is the following mathematical relationship:

$$
\begin{aligned}
h_1 &= \sigma(\omega_{11}x_1 + \omega_{12}x_2), \\
h_2 &= \sigma(\omega_{21}x_1 + \omega_{22}x_2), \\
y &= \sigma(v_1h_1 + v_2h_2).
\end{aligned}
$$

Firstly, calculate the partial derivative of the error $\frac{\partial L}{\partial y} = (y-t)$ in the output layer. Secondly, the error of the output layer is reversely pushed backward layers by layers to obtain the partial derivative $\frac{\partial L}{\partial h_i}$ of the total error to each layer, given as:

$$\frac{\partial L}{\partial v_1} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot h_1,$$
$$\frac{\partial L}{\partial v_2} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot h_2,$$
$$\frac{\partial L}{\partial h_1} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot v_1,$$
$$\frac{\partial L}{\partial h_2} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot v_2.$$

Then, use $\frac{\partial L}{\partial h_i}$ of the node, to calculate the partial derivatives $\frac{\partial L}{\partial \omega_i}$ of the corresponding input parameters w and b. Finally, use the obtained partial derivatives $\frac{\partial L}{\partial \omega_i}$ to adjust the corresponding parameters w and b, given as:

$$\frac{\partial L}{\partial \omega_{11}} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot v_1\cdot\sigma'(\omega_{11}x_1+\omega_{12}x_2)\cdot x_1,$$
$$\frac{\partial L}{\partial \omega_{12}} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot v_1\cdot\sigma'(\omega_{11}x_1+\omega_{12}x_2)\cdot x_2,$$
$$\frac{\partial L}{\partial \omega_{21}} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot v_2\cdot\sigma'(\omega_{21}x_1+\omega_{22}x_2)\cdot x_1,$$
$$\frac{\partial L}{\partial \omega_{22}} = (y-t)\cdot\sigma'(v_1h_1+v_2h_2)\cdot v_2\cdot\sigma'(\omega_{21}x_1+\omega_{22}x_2)\cdot x_2.$$

2.1.4 Universal Approximation Theorem

According to the Universal Approximation Theorem, a feed-forward neural network has a linear output layer and at least one hidden layer with any activation function. As long as the network is given sufficient hidden units, it can approximate any function from one finite-dimensional space to another with arbitrary precision [3]. An MLP with at least one hidden layer with an activation function can fit any function.

For example, a function passes through a sigmoid, given as:

$$\sigma(x) = \frac{1}{1+e^{\omega x+\omega_0}} + b,$$

where b determines the different intercepts, resulting in a y-axis shift. ω_0 determines the shift of x axis. Similarly, w determines the slope of the line, thus affecting the slope of the sigmoid. As Figure 2.5 shown, we can combine two sigmoid functions approximately into a step function.

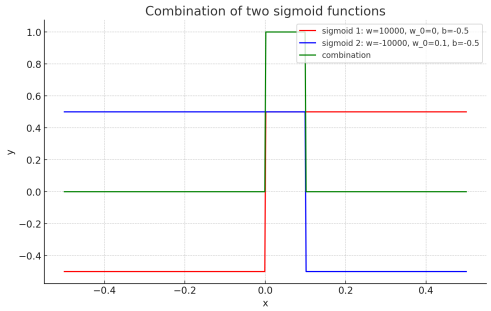

Figure 2.5: An example of the combination of two sigmoid functions. The combination approximates a step function when the parameter ω becomes large enough.

From Figure 2.5 (b), we can approximate any function as long as we have enough step functions and carefully tune the weights they add together. This process is similar to approximation by piecewise constant function.

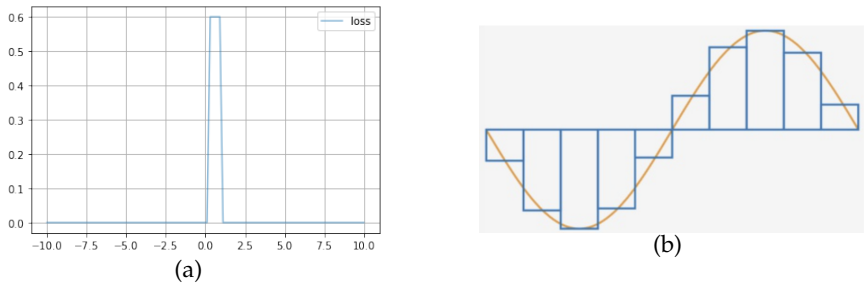

Figure 2.6: (a) A step function generated by two sigmoid function. (b) Enough neurals which contained enough sigmoid function can generate enough sigmoid functions. We can approximate any function by summing all sigmoid functions.

2.2 Convolutional Neural Networks (CNNs)

Convolutional neural networks are the most widely used deep neural networks. They have achieved the best current results in many problems in machine vision, in addition to their successful applications in natural language processing, computer graphics, and other fields. Because CNNs can reduce the number of parameters and capture local features and hierarchical relationships in datasets.

2.2.1 Components of CNNs

Convolutional Kernels

The CNN is constructed using several components. The first one is the convolutional kernel. The convolution operation is a mathematical operation.

The convolution of functions f and g donates $(f * g)$. In deep learning, the convolution operation is one function (convolution kernel) sliding over another (tensor data) function. Then, the dot multiplication is calculated at each position and summed to get a new tensor. From Figure 2.7, take single-channel convolution as an example. The shape of the input tensor data is $(1,\ 3,\ 3)$, which means the tensor has one channel, width is 3, and height is 3. Assume the convolution kernel size is $(2,\ 2)$. After this convolution operation, the shape of the output tensor becomes $(2,\ 2)$.

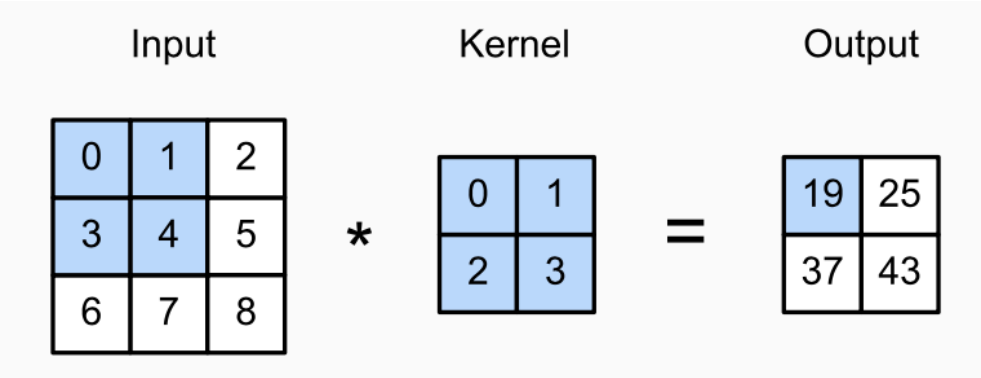

Figure 2.7: An example of convolution operation [4]. The shaded portion is the first part of the convolution operation. It is calculated as $0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$. Then, shift the convolution kernel and compute other part.

As for multi-channel convolution, assuming that there is two convolution kernel and two tensor data, the shape of the convolution kernel will become $(2,\ 2,\ 2)$ and the shape of the tensor dataset will become $(2,\ 3,\ 3)$. The convolution operation is as Figure 2.8. The pixel value of each channel is convolved with the value of the corresponding convolution kernel channel so that each channel will correspond to an output convolution result, and the two convolution results are summed up at the corresponding positions to get the final convolution result.

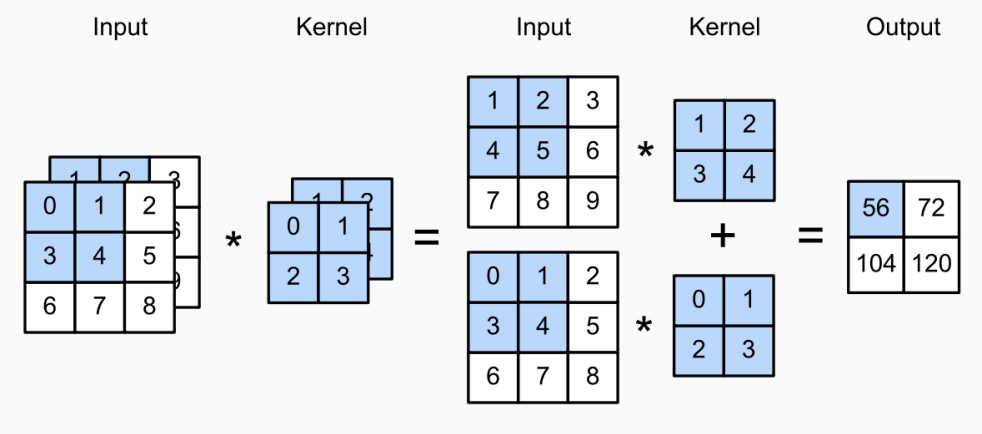

Figure 2.8: An example of one convolution layer with two channels [4]. Two convolution results are summed up at the corresponding positions to get the final convolution result.

Padding Operation

In CNNs, padding is also a commonly used technique. When a convolutional operation is performed, less data at the edges of the tensor dataset is computed by the convolutional kernel. Thus, information will be lost at the edges of the tensor. To solve this problem, we can add extra pixels to the edges of the image, which is called padding. Another purpose of padding is to control the size of the feature maps. Without padding, the convolution operation reduces the size of the feature maps. Therefore, we cannot design a deep CNN because the feature map size will finally become zero. By padding, we can create a deeper CNN. There are two common padding methods: zero padding and reflect padding.

Zero padding means adding pixels with a value of 0 to the edges of the image. The advantage of zero padding is that no new information will be added to the tensor dataset. It does not distort the information of the original image. From Figure 2.9, the shape of the dataset becomes $(1,\ 4,\ 4)$ after zero padding, and the output feature map has more dimensions than the output without padding. In this case, we save more information after the convolution operation. The reflect padding means adding pixel values from the edges of the image into the padding pixels. The advantage is to create a more natural extension, which can benefit from dealing with images with continuous characteristics.

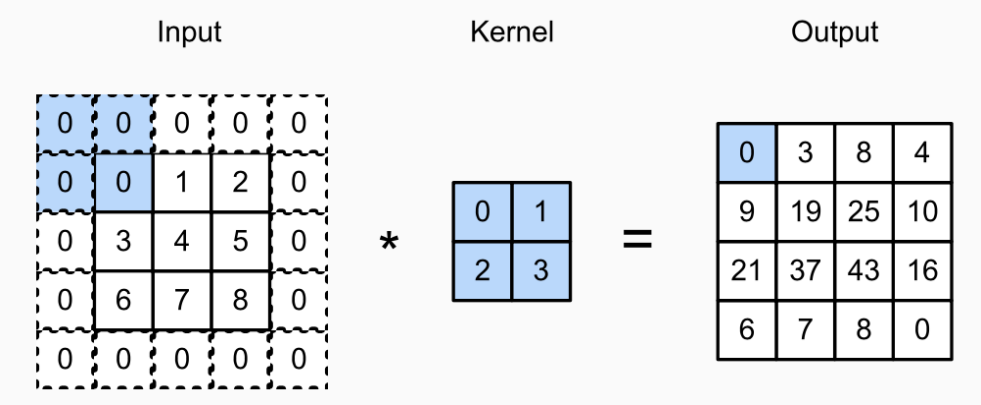

Figure 2.9: An example of zero padding technique [4]. The shape of the dataset become $(1, 4, 4)$ after zero padding, and the output feature map is larger.

Striding Operation

In CNNs, the striding is the distance which the convolutional kernel slides over the tensor dataset. A one stride means the kernel moves one frame at a time, a two stride means the kernel moves two frames at a time, and so on. A convolution operation with 1 stride can extract more information but is more expensive in computing, while a convolution operation with 2 stride (or larger) can reduce the computational complexity.

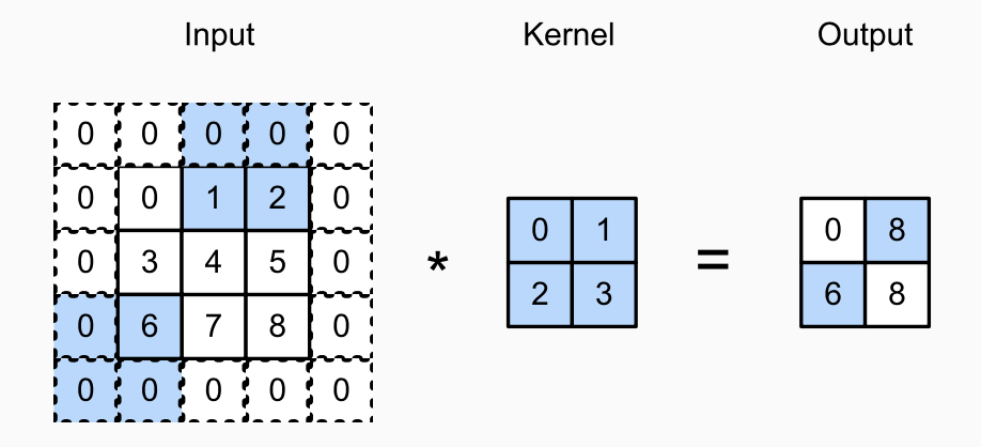

Figure 2.10: An example of a convolution operation with 2 stride [4]. It can reduce the feature map from $(3, 3)$ to $(2, 2)$.

Batch Normalisation

The second component for CNN is the Batch Normalisation (BN). The BN is the normalization of the input to the network such that the data's distri-

bution has a mean of 0 and a standard deviation of 1. Learning the model parameters depends on the input data, and BN normalizes the data to have a smoother distribution. It is one solution to the problem of unbalanced data distribution in some dimensions, concentrated in some dimensions and loose in others. Thus, the learning of the parameters is also more balanced, and the extreme value distribution of the parameter norm is solved. It is equivalent to regularizing the parameters, thus having some effect on the overfitting problem.

Take the two-dimensional parameter space as an example. As shown in Figure 2.11, when BN is not used, the difference in features between the two dimensions is large, and the strength of x_1 is significantly weaker than that of x_2. So the parameter ω_1 and ω_2 require differences in adaptation characteristics, which means the value of w_1 becomes larger than w_2. The results are easily dominated by ω_1. If x_1 is slightly larger in the test dataset, and over-fitting phenomenon occurs. After using BN, the range of features in the two dimensions is close, and the update of parameters ω_1 and ω_2 is relatively balanced, which is equivalent to using a regularization for the parameters. In this case, the results are not easily affected by the dominance of a certain dimensional feature, and the over-fitting is solved.

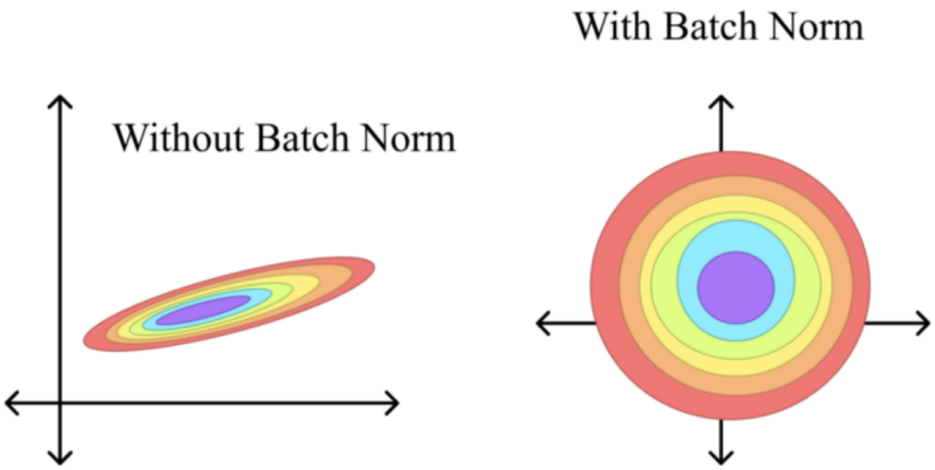

Figure 2.11: A Two-dimensional Example of BN [5]. The unbalanced data distribution happens without BN.

Pooling layers

The third component for CNN is the Pooling layer. The pooling operation follows the general convolution operation. In essence, the pooling operation is sampling, where the input feature maps are compressed in some way to

speed up the computation. As shown in figure 2.12, it indicates that the values in the neighborhood of a 4×4 feature map are scanned with a 2×2 filter with a step size of 2. In this case, the maximum value is selected for output to the next layer, called Max Pooling.

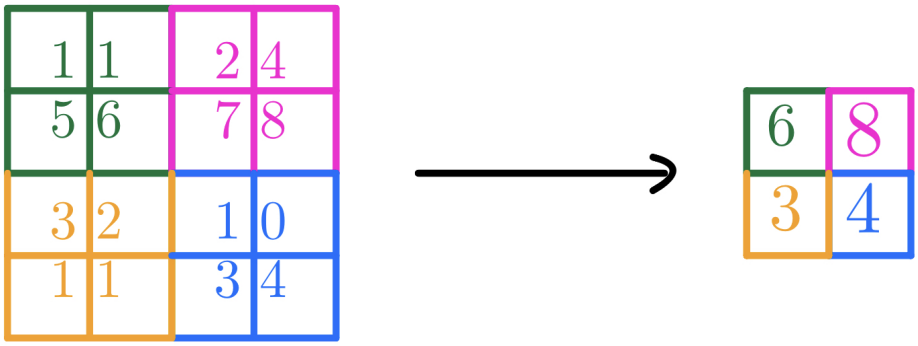

Figure 2.12: An Example of Maximum Pooling. The pooling operation follows the general convolution operation. Replace the convolution kernel with the maximum operation filter.

In addition to the maximum pooling layer, average pooling is also common. Pooling prevents the overfitting of the model, which is because the image's maximum features (or average features) can be mentioned to represent the image instead of sticking to all the details using pooling methods. Pooling can also reduce the computational effort of the model and speed up the inference efficiency of the model.

2.2.2 Inventions of CNNs

The first genuinely convolutional neural network was proposed by LeCun in 1989 and later improved [6]. It was used for handwritten character recognition and originated from various current deep convolutional neural networks. The network in the literature consists of convolutional and fully connected layers. The network's input is a (16, 16) normalized image, and the output is ten classes from 0 to 9, with three hidden layers between input and output. The structure of this network is shown in Figure 2.13.

This paper proposes the concepts of weight sharing and feature maps, which are used to this day and are the prototypes of convolutional layers. The network consists of one input layer, one output layer, and three hidden layers, where the hidden layers H_1 and H_2 are two convolutional layers and H_3 is a fully-connected layer. The activation function of the network is selected as the *tanh* (hyperbolic tangent) function, and the loss function is selected as the

mean squared error function, which is the mean value of Euclidean distance. The weights of the network are initialized with uniformly distributed random numbers, the backpropagation algorithm is used to calculate the parameter gradient values during training, and the online stochastic gradient descent method is used to update the gradient values.

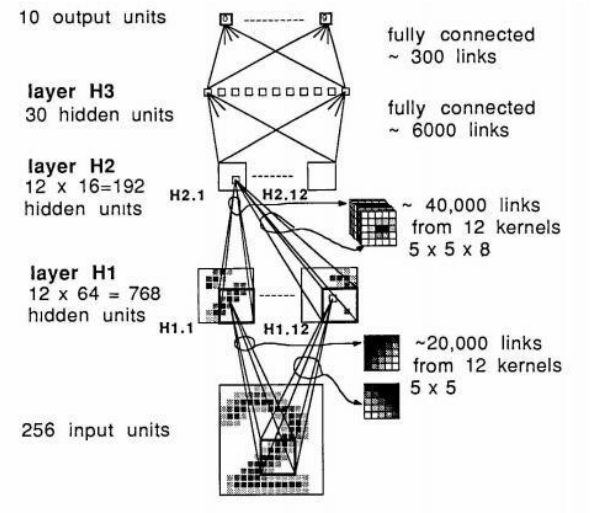

Figure 2.13: The LeCun's CNN structure [6]. It is the first CNN and used for the classification of hand-writing numbers.

2.2.3 Advantages of CNN

The CNN layers share some parameters, which means they require far fewer parameters than a fully-connected MLP. For example, each convolutional kernel convolves over the entire input, significantly reducing the number of parameters.

The CNNs can capture local features and hierarchical relationships in space. Convolutional layers can be seen as automatic feature extractors that capture essential features in an image (e.g., edges, corner points, and so on) [7]. MLPs, on the other hand, often require manual extraction of image features.

The CNNs can better handle high-dimensional data. For example, when processing image or video data, CNN can directly accept 3D (height, width, color channel) or higher dimensional data as input. MLP needs to flatten these data into one-dimensional data.

2.2.4 CNNs for Image Classification Tasks

The classification task is to assign a label to an image from a given set. As Figure 2.14 shown, it is an example of CNN for classification tasks. The first

two layers are the convolutional layers, which extract features from images. The last two fully-connected layers perform weighted summations of the output of the features from the previous layer and input the results into the activation function to finally complete the classification of the target. The reason why multiple fully-connected layers are needed is that the dimensionality of a vector will probably be higher after the flatten operation. However, for the classification task, a lower dimensionality is usually needed. Usually, 2 or 3 fully-connected layers are used to eliminate this drastic change, and the dimensionality of the fully-connected layers is gradually reduced.

In CNN, we need to increase the channels when the number of layers increases. Each channel learns a specific feature in a CNN. In the shallow layers, the neural network mainly learns basic information such as edge information, geometric information, color information, and so on, which has been proven to be the same no matter whatever the computer vision task is [7]. The amount of basic information (horizontal, vertical, diagonal lines, small arcs, etc.) is insignificant. Thus, smaller channels can be enough, so we need small receptive fields. As the network deepens, the features learned become more abstract and specific to the specific task for which the network needs learning. Therefore, the number of channels must increase to make the network more expressive and cover as many essential features as possible, so we need larger receptive fields.

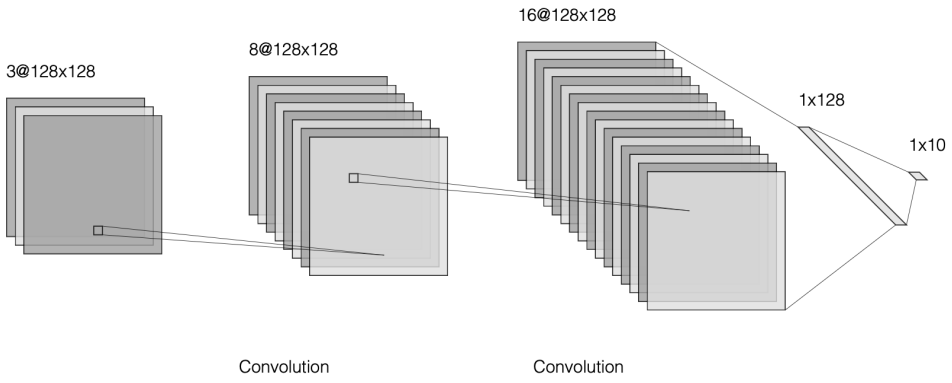

Figure 2.14: An example of CNN for classification tasks [8]. There are 2 convolutional layers in this networks. After convolution operations to extract information, use fully-connected MLP to classify features.

2.3 Residual neural networks (ResNets)

Since the great success of CNNs for classification problems on ImageNet, researchers have found that model accuracy does not always improve with network depth [9]. The ResNets is a solution to build deeper CNNs. Because ResNet structures can avoid Vanishing (Exploding) Gradient problems.

2.3.1 Structure of ResNets

The residual network blocks are regular neural networks with identity mappings linked between layers. The identity mapping refers to: for any set A, if the mapping $f : A \to A$ is defined as $f(a) = a$. Each element a of A is defined to correspond to itself. Then, f is an identical mapping on A.

The jth layer of a normal network has an input Y_j and wants to output $G(Y_j, \theta_j)$. In the deep learning field, identity mapping refers to a mechanism that directly forwards the input to the output. As for ResNets, the output of the jth layer is superimposed directly onto the output of the $(j+1)$th layer neural network. Thus, the output of a ResNet block is $H(Y_j, \theta_j) = G(Y_j, \theta_j) + Y_j$, and its structure is shown in the Figure 2.15.

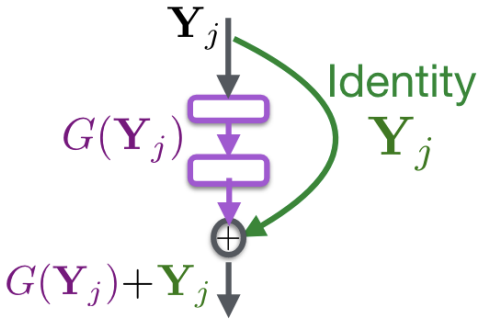

Figure 2.15: The structure of a ResNet block [10]. The output of this ResNet block is the output of a neural network in addition with an identity mapping.

2.3.2 The Inventions of ResNets

Degradation in Deep Networks

Intuitively, it is easy to conclude that increasing the network depth allows the network to perform more complex feature extraction, and therefore deeper models can achieve better results. However, the facts are the contrary. It has been found that the model accuracy only sometimes improves as the network depth increases. According to K. He's paper, the error rate of the 56-layer network is higher than that of the 20-layer one, both on the training set and the test set [9].

The first 20 layers are theoretically sufficient to meet our performance requirements. The new layers are somewhat redundant. For these new layers to achieve 100% performance, increasing the training time exponentially is mandatory to achieve model convergence. When a 56-layer neural network fails to converge fully, its results are worse than those of a 20-layer network. This phenomenon is named degradation[9].

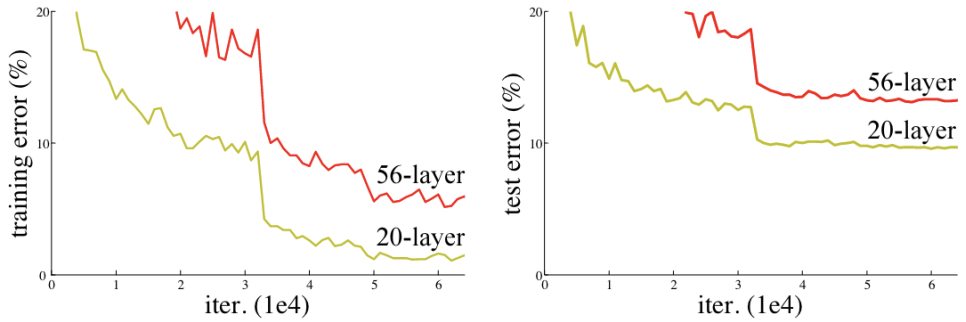

Figure 2.16: K. He's result [9]. There are higher error rates of the 56-layer network than the 20-layer, both on the training set and on the test set.

Reason for Degradation

The reason for degradation can be described from functional analysis. Each neural network can be treated as a function. Figure 2.17 (a) illustate neural networks F_1, F_2, ... , F_6 with the increasing number of layers. The colored regions represent the approximated function spaces. The F_2 is a deeper-layer neural network than F_1, and F_3 is deeper than F_2, etc.

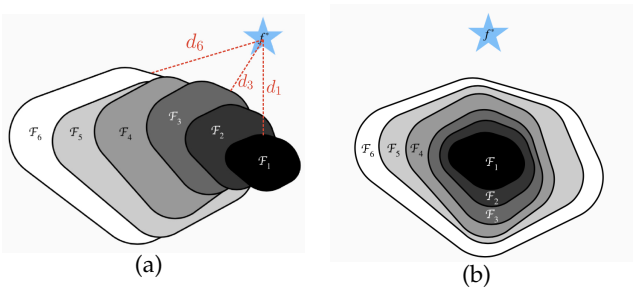

Figure 2.17: The function spaces of non-nested and nested functions [4]. In non-nested functions, the distance (d_3) between the function space generated by F_3 and ground truth f^* is shorter than d_6, which means the deeper neural networks cannot bring a higher accuracy. But nested functions can.

According to Chapter 2.1.4, Universal Approximation Theorem, a deeper

network means approximating more parameters inside, resulting in a larger function space. However, as Figure 2.17 (a) shows, the distance (d_3) between the function space generated by F_3 and ground truth f^* is shorter than d_6, which shows that a larger space does not guarantee a shorter distance to the ground truth f^* [11], which means deeper neural networks cannot always bring a higher accuracy. In order to solve the above problem, the function space induced by the $(j+1)$th layer network must topologically cover the space of the jth layer, which identity mappings can achieve. In this case, those networks are considered functions nested to others, as shown in Figure 2.17 (b).

Types and Applications of ResNets

In K. He's ResNets, the 20-layer network is a subset of the 56-layer network. So the solution space of the 56-layer network contains the solution space of the 20-layer network. If an identity mapping is added from layer 21 to layer 56, the 56-layer network is equivalent to a 20-layer standard network and a 36-layer network which needs learning the residual. It ensures that this 56-layer network is at least as effective as the original 20-layer network for similar training epochs. Moreover, K. He proposed a general architecture of ResNet for the image classification problem, which is ResNet-34. There are 34 layers inside ResNet-34. The first layer is an input layer, and the final layer is fully-connected. In the middle are 16 groups of residual modules, and the channel size rises gradually from 64 to 512. By increasing the channel size, CNN can learn more complex and abstract feature representations since each channel can be considered a filter responsible for detecting specific features in the input. More channels mean that the network can learn several different types of features simultaneously, thus improving the network's ability to characterize the input data.

ResNet has proven its ability for higher accuracy of image classification. In the 2015 Object Recognition Award, researchers applied ResNet models to achieve significant performance based on ImageNet datasets, including CIFAR-10 and CIFAR-100. The ResNets stand out among several convolutional neural networks, such as VGG and GoogleNet.

In the experiments of classification of lung pathology, ResNet-50 has a correct rate of 90.37% on the test set. Moreover, it is much higher than the 60.33% accuracy of Artificial Neural Network (ANN) - based classifiers and traditional machine learning algorithms such as SVM algorithm [12].

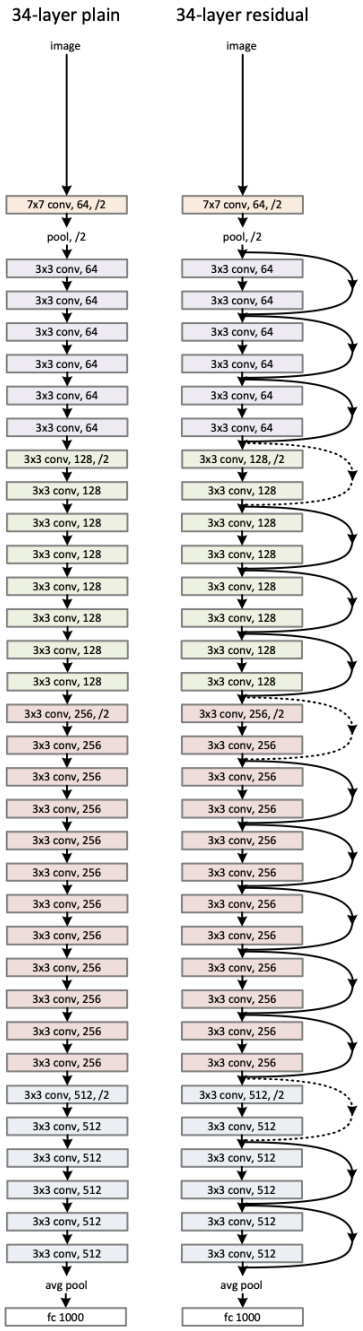

Figure 2.18: ResNet-34 [9]. ResNet-34 is the first ResNet proposed by K. He.

2.3.3 Avoid Vanishing Gradients

Feature Extraction of deep CNNs

We can build deeper networks to accomplish more complex tasks thanks to the development of deep neural networks. For instance, classification tasks are solved by deep CNNs, and regression tasks of series data are solved by deep recurrent networks (RNNs). According to the Universal Approximation Theorem, a neural network with one hidden layer can theoretically approximate arbitrary continuous functions to infinitely minor errors [13].

However, deep-layer neural networks with multiple hidden layers can often learn more complex patterns in the data than a single hidden-layer neural network. The multi-level abstraction learned from deep neural networks to fit functions is more potent than single-hidden layer neural networks. Each hidden layer may learn to extract different features of the input data. Usually, the first hidden layer may learn only basic features, such as edges and basic shapes. Deeper layers may learn higher-level features, such as some features from a part of the data. The higher-level features are learned from the lower-level features. Higher-level features will be more abstract and more useful for prediction. For example, a CNN is trained to classify images of animals. The first hidden layer might learn to recognize edges and basic shapes in the image, such as ovals and cones. The next hidden layer may learn to recognize patterns composed of these shapes. For example, ovals and cones may form the tail fin of a fish. Higher layers may learn to recognize specific animals based on combinations of these lower-layer feature arrangements, such as fish, dogs, and rabbits.

Reason For Vanishing Gradients

When the number of neural network layers is too deep, the vanishing gradients will happen. The backpropagation in deep neural networks is the reason for vanishing gradients[14]. The current approach to optimizing neural networks is to calculate the gradient of the loss function by backpropagation to guide the optimization of updating the weights of the deep network. The deep network consists of many nonlinear layers stacked on each other, and each nonlinear layer can be considered a nonlinear function. Thus, the whole depth network can be considered a composite nonlinear multivariate function.

Take the structure of Figure 2.12 as an example. In standard neural networks, the derivative of the loss function is:

$$\frac{\partial Y_{j+1}}{\partial Y_j} = \frac{\partial Y_{j+1}}{\partial H_1}\frac{\partial H_1}{\partial H_2}\frac{\partial H_2}{\partial Y_j}.$$

If the gradient of each layer is less than 1, the total gradient will decay exponentially as the number of layers increases, called the vanishing gradient. But after adding the residual layer, the new gradient becomes

$$\frac{\partial Y_{j+1}}{\partial Y_j} = \frac{\partial}{\partial Y_j} G(Y_j,\ \theta_j) + 1.$$

The 1 in this formula of new gradient indicates that the short-circuiting mechanism propagates the gradient losslessly. The other residual gradient needs to pass through the layer with weights. Even if the residual gradient is smaller, the presence of one does not cause the total gradient to vanish.

2.3.4 Bottlenecks of ResNets

The first potential bottleneck of ResNets is their complexity. As residual modules are created and applied, the number of layers of a neural network increases geometrically. The number of required parameters and computations also increases, which makes training and inference slower and more resource-intensive.

Another potential bottleneck for ResNets is the risk of overfitting. Overfitting occurs when ResNets learn patterns specific to the training data but cannot be generalized to unseen data. It can lead to poor performance on test or validation sets, which can be mitigated by regularization or adding sample data [15].

Chapter 3

Ordinary Differetial Equations (ODEs)

In this chapter, we aim to provide an overview of Ordinary Differetial Equations, including initial-value problem, ODE Systems, and stability of ODEs. These three parts will be discussed with clear explanations and illustrative examples. Also, we will explain the numerical menthods of ODEs.

3.1 Introduction to ODEs

ODEs are traditional mathematical modeling methods that are important tools for understanding and predicting systems in physics, chemistry, and biology and are widely used in these fields. For example, ODEs are used to model a wide range of phenomena, including physical laws, chemical changes, and biological processes.

3.1.1 Standard ODEs

Mathematical Definiation of ODEs

An ordinary differential equation (ODE) is a mathematical equation that describes how the time of an independent variable affects the change in one or more dependent variables. It usually involves one or more variables and their derivatives. The mathematical form of the simplest ODE is:

$$\frac{dy}{dt} = f(t,\ y)$$

where t is the time variable, y is the dependent variable, and f is the function describing the change. The derivative $\frac{dy}{dt}$ represents the rate of change of y with respect to t.

Classification of ODEs

The ODEs can be classified by orders. The highest derivative in this equation determines the order of an ODE. For example, $\frac{dy}{dx} + p(x)y = g(x)$ is a first-order ODE, where p(x) and g(x) are given functions of x and y is the function to be found. Also, the ODEs can be classified as linear and non-linear. A linear ODE is an equation in which the unknown function and its derivatives are linearly related. $\frac{dy}{dx} + p(x)y = g(x)$ is also a linear ODE. An equation is a non-linear ODE if the unknown function in the equation or its derivative appears in a non-linear fashion, which may include the product of y or its derivative, etc. For example, $\frac{dy}{dx} = y^2$.

3.1.2 The initial-value problems (IVPs)

The IVP problems are ODEs that require not only to find a solution that satisfies the differential equation, but also the given initial condition at a certain point. For example, the IVP problem of a first-order ODE can be expressed as:

$$\frac{dy}{dt} = f(t,\ y),\ \ y(t_0) = y_0$$

where the f is a function of t and y. The t_0 and y_0 are initial statements.

3.2 ODE Systems

ODE systems often arise when we build mathematical models for complex physics, engineering, or biology problems, such as the Susceptible-Infected-Recovered (SIR) model.

3.2.1 Definition of ODE Systems

An ODE system is a collection of differential equations involving the same independent variable. These equations are coupled, meaning that the solution to one equation depends on the solution to the others. For example, the SIR model describes the dynamics of infectious diseases in a closed population [16], as Figure 3.1 shown.

$$\frac{dS}{dt} = -\beta SI,\ \ \frac{dI}{dt} = \beta SI - \gamma I,\ \ \frac{dR}{dt} = \gamma I$$

Here, S is the susceptible people, I is the infected people, and R is the recovered people. The β is the transmission rate of the disease, and γ is the recovery rate. The first equation describes the rate of change of susceptible

individuals. The second equation describes the rate of change of infected individuals. Moreover, the third equation describes the rate of change of recovered individuals.

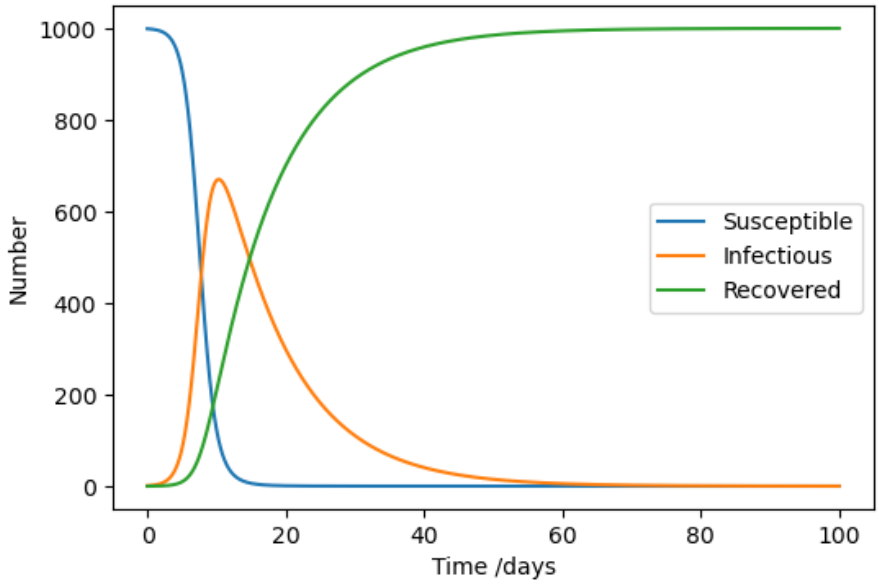

Figure 3.1: An Example of SIR Model. At the beginning of this simulation, there are 1000 susceptible people in a closed population. As time progresses, the number of infected people increases and then decreases.

3.2.2 Transformation Of Higher-order ODEs

The higher-order ODEs are ODEs that have second or higher-order derivatives. For example, the second-order ODEs is:

$$\frac{d^2y}{dt^2} = f\left(t,\ y,\ \frac{dy}{dt}\right)$$

The order of an ODE is the highest-order derivative's order. The higher-order ODEs can describe more complex problems than the first-order ODEs. The whole higher-order ODEs can be transform to a first-order ODE system, and here is the proof. Consider a general $n-th$ order ODE:

$$\frac{d^ny}{dt^n} = f\left(t,\ y,\ \frac{dy}{dt},\ \frac{d^2y}{dt^2},\ \ldots,\ \frac{d^{n-1}y}{dt^{n-1}}\right)$$

We can introduce new variables to convert this into a system of first-order ODEs, like:

$$y_1 = y$$

$$y_2 = \frac{dy}{dt} = y_1'$$

$$y_3 = \frac{d^2y}{dt^2} = y_2'$$

$$\vdots$$

$$y_n = \frac{d^ny}{dt^n} = y_{n-1}'$$

Now, we have a system of first-order differential equations:

$$y_1' = y_2$$

$$y_2' = y_3$$

$$\vdots$$

$$y_{n-1}' = y_n$$

$$y_n' = f(t, y_1, y_2, \ldots, y_n)$$

This system of first-order ODEs is equivalent to the original $n - th$ order ODE, and any solution to this system is also a solution to the original equation.

3.3 Stability of ODEs

The stability of ODEs describes the effect of small perturbations or changes in initial conditions on the solution of the equation. Mathematically, the equilibrium solution y^* is stable if, for $\forall \varepsilon > 0$, there exists a $\sigma > 0$ such that if the initial condition $y(0)$ satisfies $|y(0) - y^*| < \sigma$, then for all $t \geq 0$, $|y(t) - y^*| < \varepsilon$. For example, consider the simple first-order ODE:

$$\frac{dy}{dt} = \lambda y, \quad with\ \lambda < 0.$$

The solution to this ODE is $y(t) = Ce^{\lambda t}$, $with\ \lambda < 0$, where C is a constant determined by the initial condition. For any initial condition, as $t \to \infty$, $y(t) \to 0$. Hence, this system is stable. From Figure 3.2, for example, ODE $\frac{dy_1}{dt} = -y_1$, $\frac{dy_2}{dt} = -y_2$ is stable.

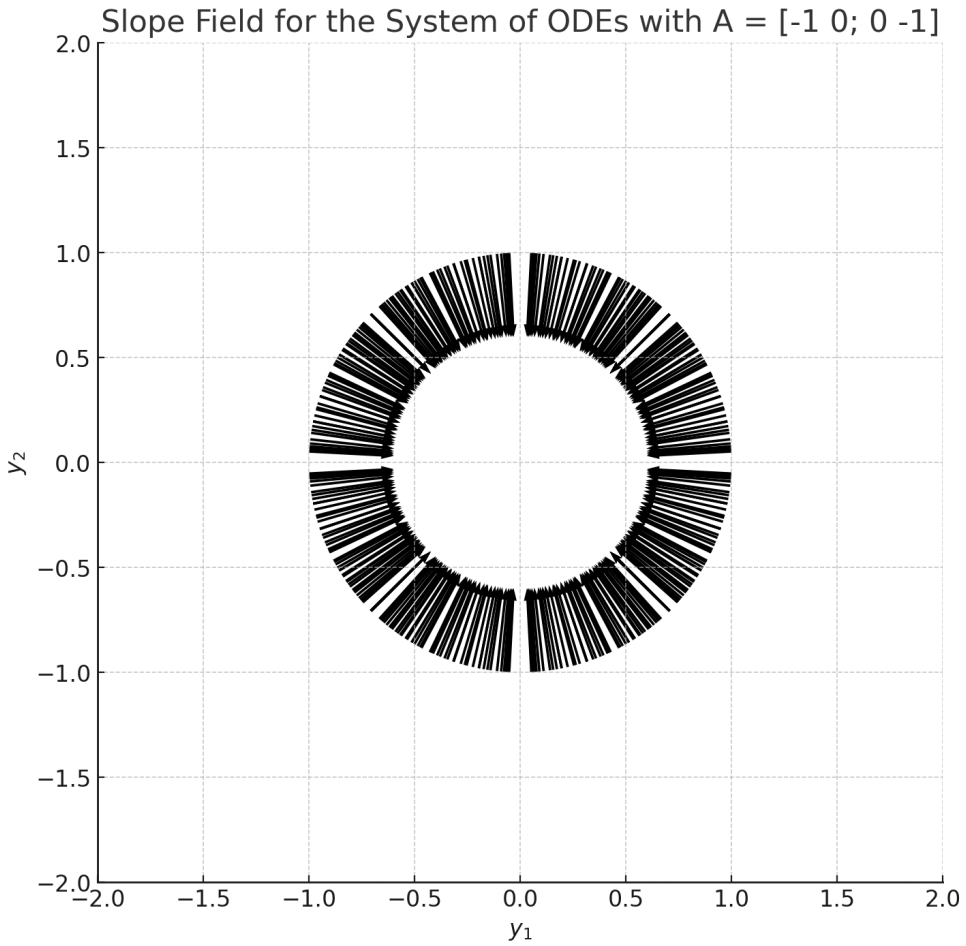

Figure 3.2: An example of a stable ODE $\frac{dy_1}{dt} = -y_1, \ \ \frac{dy_2}{dt} = -y_2$. Where all $\rightarrow$ can direct to the zero point, it is a stable ODE.

As for the same ODE:

$$\frac{dy}{dt} = \lambda y, \ \ with \ \lambda > 0.$$

The solution to this ODE is $y(t) = Ce^{\lambda t}, \ \ with \ \lambda > 0$, where C is a constant determined by the initial condition. For any non-zero initial condition, as $t \rightarrow \infty$, $y(t) \rightarrow \infty$. Hence, this system is unstable. From Figure 3.3, for example, ODE $\frac{dy_1}{dt} = y_1, \ \ \frac{dy_2}{dt} = y_2$ is unstable.

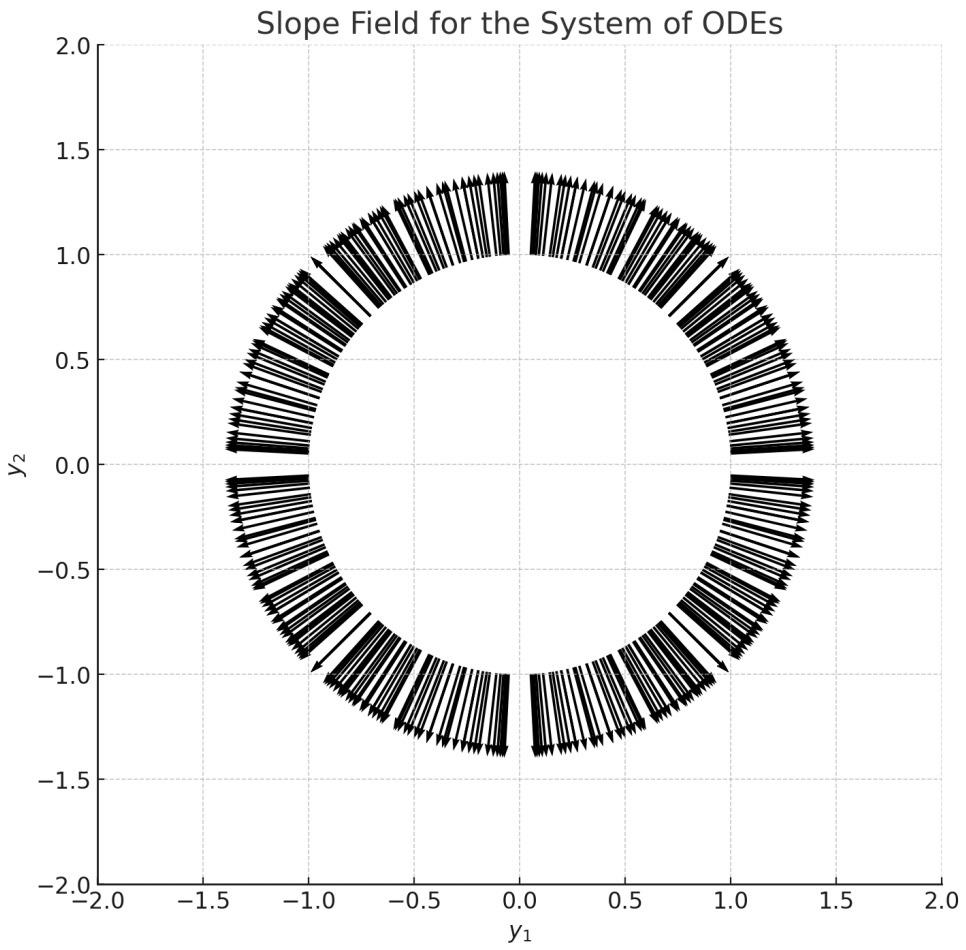

Figure 3.3: An example of an unstable ODE $\frac{dy_1}{dt} = y_1$, $\frac{dy_2}{dt} = y_2$. Where all $\rightarrow$ cannot direct to the zero point, it is an unstable ODE.

3.4 Numerical Solution of ODEs On Computers

The solution to the ODE is approximated by discretizing the equation and solving it at a series of time points, with numerical methods. In the rest of this section, we introduce some of the most popular methods, namely Euler's methods and Runge-Kutta's methods.

3.4.1 Euler's method

Taylor Series

The Taylor series is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point. The Taylor series of a function $f(x)$ about a point a is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \frac{f'''(a)(x-a)^3}{3!} + \cdots$$

Here, $f'(a)$, $f''(a)$, $f'''(a)$... are the first, second, third derivatives of f evaluated at a etc. And $n!$ denotes the factorial of n. The Taylor series provides a way to approximate any function as a polynomial. The more terms you include in the series, the better the approximation becomes. In many cases, only the first few terms of the series are used, which provides a good approximation near the point a.

From Taylor Series to Euler's Methods

Consider an IVP:

$$\frac{dy}{dx} = f(x,\ y),\ \ y(x_0) = y_0.$$

Given a small step size h, the solution $y(x)$ at the point $x = x_0 + h$ can be approximated using the Taylor series expansion as:

$$y(x_0+h) = y(x_0) + h \cdot f(x_0,\ \ y(x_0)) + \frac{h^2}{2!} \cdot \frac{d^2y}{dx^2}\bigg|_{x=x_0} + \frac{h^3}{3!} \cdot \frac{d^3y}{dx^3}\bigg|_{x=x_0} + \cdots$$

If we truncate the Taylor series after the first term, the approximation becomes:

$$y(x_0+h) \approx y(x_0) + h \cdot f(x_0,\ \ y(x_0)).$$

which becomes a first-order method and the global error is proportional to the step size. The formula $y(x_0+h) \approx y(x_0) + h \cdot f(x_0,\ \ y(x_0))$ is Euler's method.

Advantages and Disadvantages of Euler's Methods

The Euler's methods is a linear approximation. Thus, the Euler's methods has a low computational cost. But the Euler's methods has low accuracy. Because we only use the first term to approximate the solution of ODEs. Also, because the Euler's method is iterative and errors from one step can propagate and accumulate. For example, an IVP

$$\frac{dy}{dt} = cos(t) + \lambda(y - sin(t)),$$

with the initial condition $y(0) = 0$ and $\lambda = -1$, which analytical solution is:

$$y(t) = sin(t).$$

From Figure 3.4, we obtain that there is a obvious difference between the plot of Euler's method and analytical solution for this ODE.

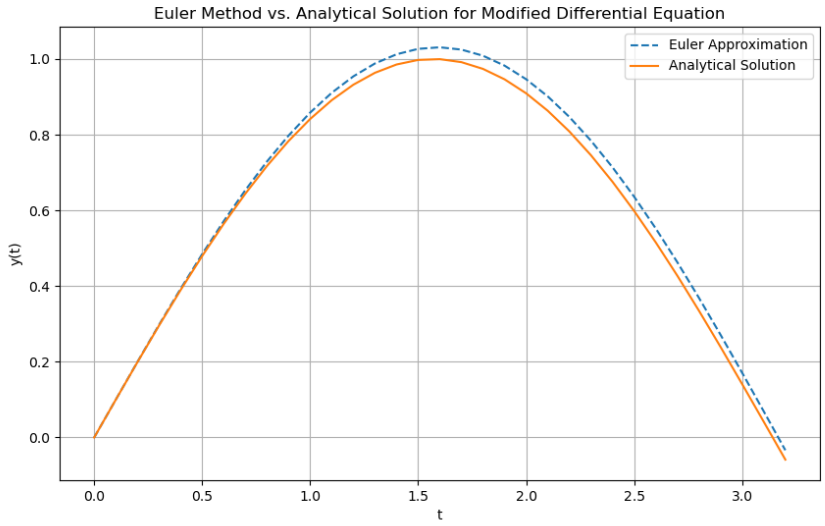

Figure 3.4: An Example of Euler's method. We can see the error between real value and numerical value increase when t increase.

3.4.2 Runge-Kutta (RK) Method

From Chapter 3.4.1, the Euler's method come from the first-term truncation of Taylor series. Consider the Taylor series expansion once more:

$$y(x_0 + h) = y(x_0) + h \cdot \frac{dy}{dx}\bigg|_{x=x_0} + \frac{h^2}{2!} \cdot \frac{d^2y}{dx^2}\bigg|_{x=x_0} + O(h^3)$$

If more terms from the Taylor series expansion truncates, the accuracy of the approximation can be enhanced. However, the computational complexity of the higher order terms in the series was too high. Thus, the method of approximating the solution by finding the weighted average of the slopes at each point in the interval was invented. The second-order Runge-Kutta method is more common and can be described as follows:

- Calculate the initial slope: $k_1 = h \cdot f(x_0,\ \ y(x_0))$.
- Calculate the midpoint slope: $k_2 = h \cdot f(x_0 + \frac{h}{2},\ \ y(x_0) + \frac{k_1}{2})$.

- Update the solution: $y(x_0 + h) \approx y(x_0) + k_2$.

Use the same IVP as an example:

$$\frac{dy}{dt} = cos(t) + \lambda(y - sin(t)),$$

with the initial condition $y(0) = 1$, which analytical solution is:

$$y(t) = sin(t).$$

From Figure 3.5, we obtain that the solution using the RK method is closer to the analytical solution compared to Euler's method. This comparison visually demonstrates the improved accuracy of the RK method over Euler's method for the same ODEs.

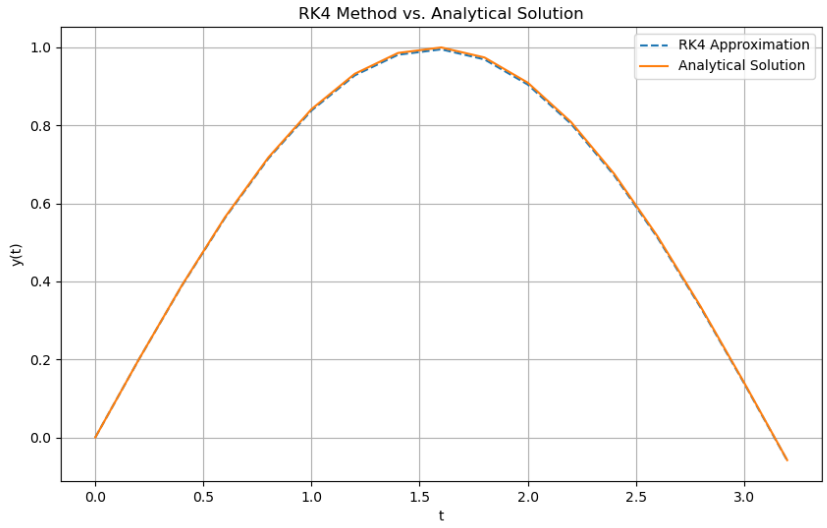

Figure 3.5: Comparison of RK and Euler's method. RK is more closed to the real answer.

3.4.3 Implicit Method and Explicit Method

Definition of Implicit and Explicit Methods

Explicit and implicit methods are two types of numerical methods used to solve differential equations. The key difference between them lies in how they compute the function's value at the next time step.

In explicit methods, the function's value at the next time step is expressed explicitly in terms of known quantities at the current and previous time steps.

This means that you can directly calculate the next value once you know the current and previous values. For example, in the Euler's method, the next value $y(t+h)$ is calculated directly from the current value $y(t)$ and its derivative $f(t,y)$ at the current time step.

$$y(t+h) = y(t) + h \cdot f(t,y)$$

In implicit methods, the function's value at the next time step is defined implicitly, meaning it appears on both sides of the equation. This means that you can't directly calculate the next value; instead, you have to solve an equation at each time step. For example, in the backward Euler method (an implicit method), the next value $y(t+h)$ is calculated using the derivative $f(t+h,\ \ y(t+h))$ at the next time step. This requires solving an equation at each step, which can be more computationally intensive.

$$y(t+h) = y(t) + h \cdot f(t+h,\ \ y(t+h))$$

Comparison of Implicit and Explicit Methods

Implicit methods are generally more stable than explicit methods. However, implicit methods are also more computationally intensive, as they require solving an equation at each time step.

For example, we'll use the simple differential equation:

$$\frac{dy}{dt} = -y, \ \ with \ \ y(0) = 1.$$

The analytical solution to this equation is an exponential decay. From the left panel of Figure 3.6 (explicit Euler), for $h = 0.1$, the solution is close to the expected exponential decay. But for $h = 2$, which exceeds the stability condition, the solution becomes unstable and oscillates. However, as for implicit Euler, for all step sizes, the solution is stable.

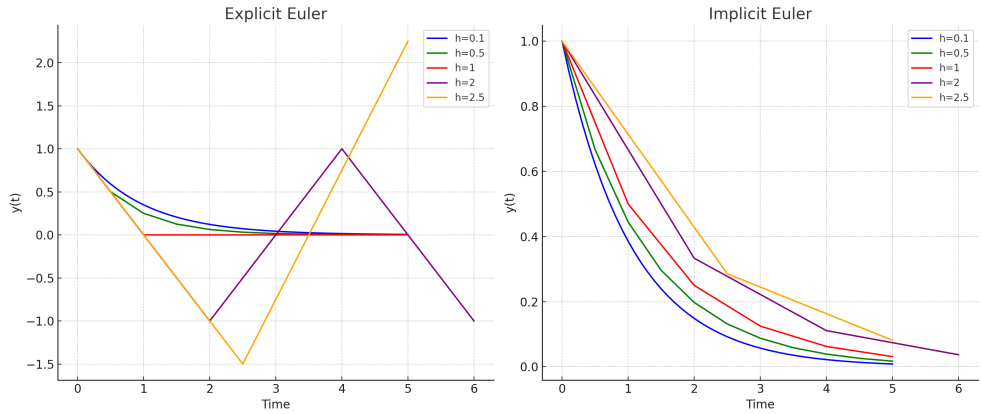

Figure 3.6: Comparison of Implicit and Explicit Methods. As for implicit Euler, for all step sizes, the solution is stable.

3.5 Autonomous and Non-autonomous ODEs

Autonomous ODEs are a type of differential equations where the rate of change of a function depends only on the current value of the function itself, without explicitly depending on time. Mathematically, an autonomous ODE can be represented as:

$$\frac{dy}{dt} = f(y),$$

where y is the unknown function of time t, and $f(y)$ is some function of y that determines how y changes over time.

Non-autonomous ODEs are differential equations where the rate of change of a function depends explicitly on time t in addition to the function itself. Mathematically, a non-autonomous ODE can be represented as:

$$\frac{dy}{dt} = f(y,\ t),$$

where y is the unknown function of time t, and $f(y,t)$ is some function of both y and t that determines how y changes over time.

3.6 Connection of ODEs with ResNets

The connection between ODEs and ResNets lies in the fact that ODEs can be used to model the dynamics of the skip connections in a ResNet. In particular, the dynamics of the skip connections can be described by an ODE in which

the dependent variable is the value of the skip connection at a given time, and the derivative of the dependent variable is the rate of change of the skip connection concerning time.

For ResNets, one residual block in Figure 1 is:

$$Y_{t+1} = Y_t + h\ F(Y_t,\ \theta_t),\ \ t \in \{0...t\}$$

For a sufficiently small h, the previous equation can be regarded as a forward Euler discretization of the initial value ODE.

$$Y'(t) = F(Y(t),\ \theta(t)),\quad Y(0) = Y_0,\ 0 \leq t \leq t$$

The connection between ODEs and ResNets is important because it allows us to use techniques from the field of ODEs to analyze and understand the behavior of ResNets. This can be particularly useful in the design and optimization of ResNets, as it allows us to use the tools and methods of ODEs to gain insight into the dynamics of the skip connections in the network and how they influence the overall behavior of the network.

3.7 ResNets with RK Method

From Chapter 3.5, the equation $Y_{j+h} - Y_j = h\ F(Y_j,\ \theta_j)$ is a representation of the forward Euler's method. From Chapter 3.4.3, the higher-order methods provide more accurate solutions than lower-order methods.

To ensure that the same computational complexity, we use the autonomous ODEs. In ResNet's structures, we can update $Y_{j+1} - Y_j = h\ F(Y_j,\ \theta_j)$ from the Euler's method (a first-order method) to the RK2 method (a second-order method). Firstly, calculate the slope at the beginning of the interval:

$$K_1 = F(Y_j,\ \theta_j).$$

Then, calculate the slope at the midpoint:

$$K_2 = F(Y_j + \frac{h}{2}K_1,\ \theta_j).$$

Finally, the midpoint slope will update the function value over the entire interval:

$$Y_{j+h} = Y_j + h\ K_2.$$

As Figure 3.7 shows, the structures of ResNets with the RK2 methods become a ResNet with two identity mappings. Theoretically, RK2 can provide a more accurate solution but also needs more computational time. We will explore

in numerical experiments if ResNets with the RK2 can increase the accuracy compared with the simple ResNet structure.

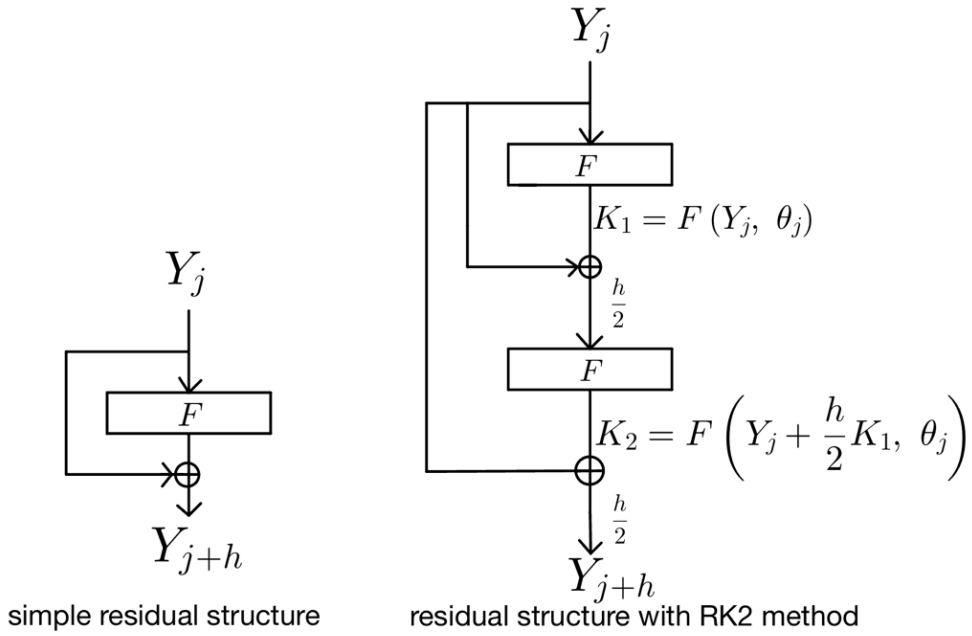

Figure 3.7: An example of the residual structure with RK2 methods. The structures of ResNets with the RK2 methods become a ResNet with two identity mappings.

In the case of autonomous ODEs, the number of neurons and the number of parameters of our new-designed ResNets with RK methods are consistent with the original ResNet. In each residual block, K_1 and K_2 are computed by a same sub-network. From Figure 3.7, both K_1 and K_2 are computed by F. In this case, our new-designed ResNets will take only a little more time to train than standard ResNets.

Chapter 4

Multi-level Methods

In this chapter, we will outline the acceleration of the residual network training process using the multilevel methods. We will present the process of multilevel methods and illustrate the theoretical validity of multilevel methods from both mathematical and deep learning directions.

4.1 Process of Multi-level Methods

The Multilevel methods can be used as a hierarchical approach to training deep neural networks. It involves training the network at various levels of complexity, gradually increasing the depth or adding more features as the training progresses. From a mathematical point of view, Section 3.5 discussed that the ResNets can be seen as ODE systems. In Chapter 2.1.2, we discussed that the training process of a neural network is to find a minimal solution to an optimization problem. Thus, combining these two sections, we can conclude that the training process of a ResNet is an optimization problem for solving an ODE system. Thus, when we divide all the residual blocks with the same parameter shape into a group, this group of residual blocks can be regarded as an ODE with a starting time point and an ending point [17].

The specific steps of the multilevel methods are that at the beginning of the training process, we use a shallow ResNet. For example, as Figure 4.1 shows, the shallow ResNet only has two residual blocks, the starting block T_0 and the end block T_1. This ResNet only contains one residual group, and the h is the step size of the explicit method for this ODE. After a few training steps, we insert a new residual block after the starting block T_0. In this way, we can obtain a deeper network, and the timestamp of the newly added residual block is $T_{0.5}$. Therefore, the explicit step size h must become half of the shallow network. With the operation of halving the explicit size to $\frac{h}{2}$, the deeper ResNet is the same ODE as the shallow one [18].

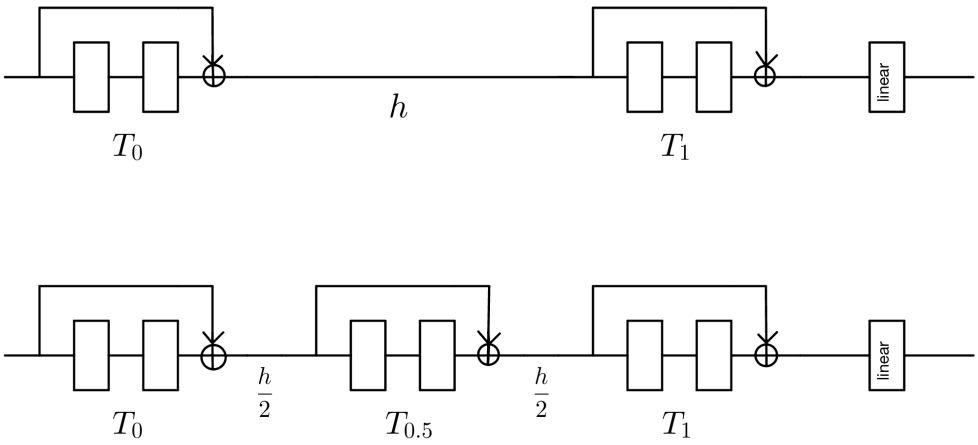

Figure 4.1: An Example of a multi-level method. We insert a new residual block to obtain a deeper network.

When interpolating from shallow to deep networks, this new residual block's weights are deep copies of the starting block, and other layers are deep copies corresponding to the same components in the shallow networks. For example, as Figure 4.2 shows, the weights of the starting block T_0 and end block T_1 in the deep network are copied from the shallow ones. And the additional residual block $T_{0.5}$ is copied from T_0. We use this interpolation approach because we consider the residual groups to be ODE systems, and the ODE systems evolve in time from start T_0 to end T_1. The additional residual block is $T_{0.5}$, theoretically inaccessible to the future value T_1.

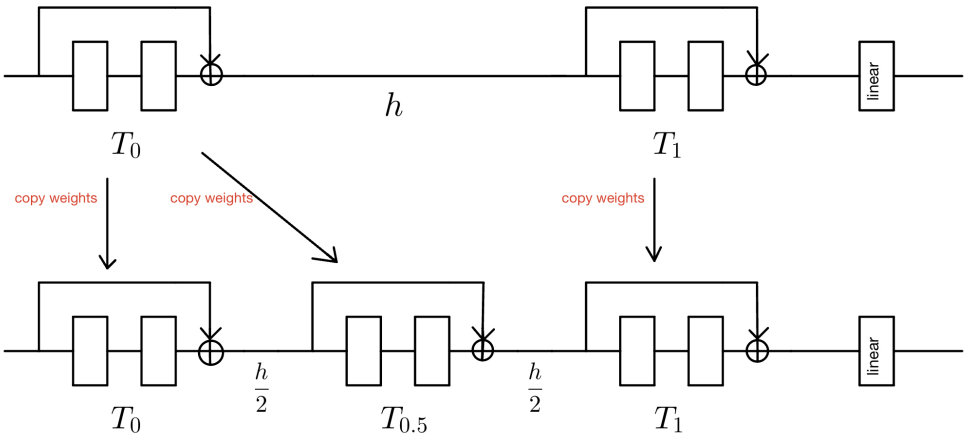

Figure 4.2: An Example of the weight copy in multi-level methods. The new residual block's weights are deep copies of the starting block.

4.2 Mathematical Background of Multi-level Methods

The mathematical background of the multi-level methods is the multi-grid methods. The multi-grid method is a numerical method used to solve partial differential equations (PDEs) problems. It is particularly effective for large-scale solving linear system of equations. When numerically solving PDEs, we usually discretize the phtsical domain of the objective function, which means to transform continuous equations into sets of discrete equations. For example, the two-dimensional Laplace equation is:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0,$$

Assume we have a uniform grid with spacing Δx in the x-direction and Δy in the y-direction. Denote the value of the function u at grid point $(i,\ j)$ as $u_{i,j}$. Then, if and only if $\Delta x = \Delta y$ and the right hand side is zero, the two-dimensional Laplace can tranform to a discrete equations:

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = 0,$$

where $i \in (0, N]$ and $j \in (0, N]$ [19].

The process of the multi-grid methods is that firstly construct an optimization problem on multiple levels, and discretise the objective function on these multiple levels. For example, assume there is an optimization problem:

$$arg\min_{x \in V} f(x),$$

where $f : V \to R$ is twice continuously differentiable objective function. From Figure 4.3, the discretisation of this formula is:

$$arg\min f_l(u_l), u_l \in V_l,$$

where $l = 1, ..., L$. The L denotes the finest resolution, and 1 denotes the coarsest resolution. Then, start to calculate this PDE on the finest level using a few iterations of a basic solver, such as the Jacobi method and the Gauss-Seidel method. Final, transfer the problem to coarser and coarser grids.

As for the more advanced multi-grid methods, it is usually more complicated to transfer the problem from different resolutions. As Figure 4.4 shown, the two most common migration methods are V-Cycle and W-Cycle. In a V-Cycle, the optimization moves from the finest mesh to the coarsest mesh and back again. In a W-Cycle, the intermediate meshes are visited several times.

The multi-level methods are based on the multi-grid methods. In multi-level method, the optimizations of objective function in deep learning from the

Chaper 2.1.3 are computed by Gradients Descent methods rather than the Jacobi method and the Gauss-Seidel method. Also, we ignore the V-Cycle and W-Cycle in multi-level method.

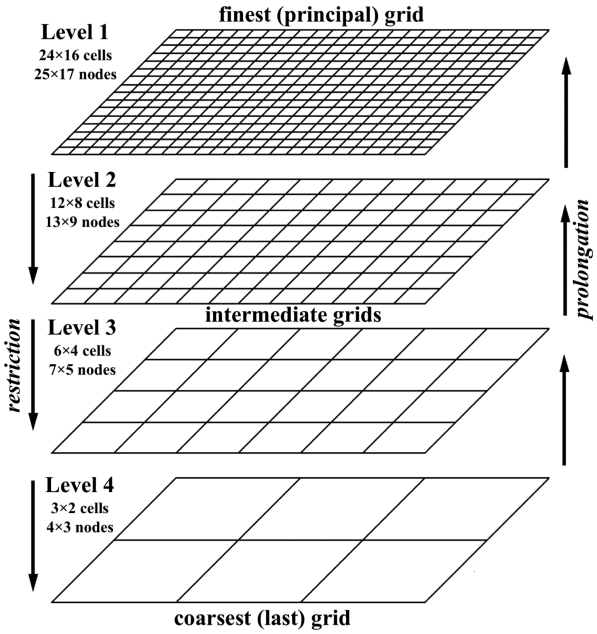

Figure 4.3: An Example of a multi-grid method. [20]. Start to calculate this PDE on the finest level, and transfer to coarser and coarser grids.

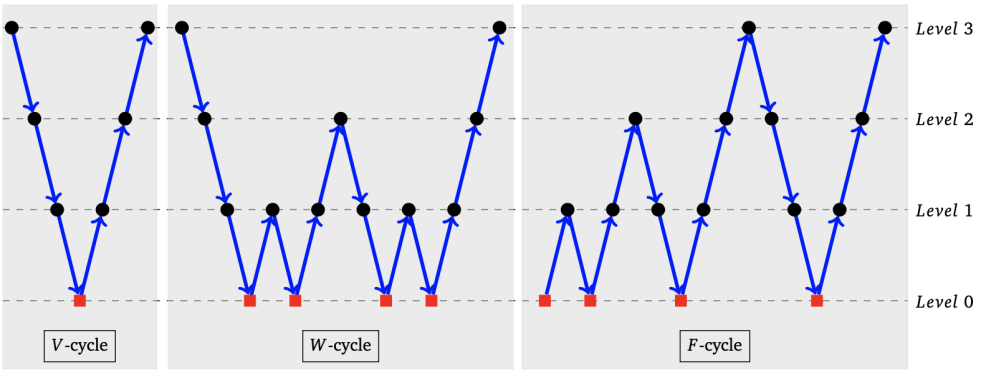

Figure 4.4: An Example of V-Cycle and W-Cycle. They are more advanced multi-grid methods, and more complicated to transfer the problem from different resolutions.

4.3 Multi-level Methods on ResNet with Multi-group Blocks

From Chapter 2.2.4, we need to increase the channel size when the number of network layers increases in CNN. We still need to follow this principle when we design ResNets. The reason is the same with CNN design: to ensure that deeper layers in a ResNet have a large enough receptive field to learn more abstract and complex patterns. The difference between CNNs and ResNets is that ResNet's channel size generally increases with the number of residual blocks rather than positively correlating with the layers. In this case, there are multiple residual groups. For example, as Figure 4.3 shows, the ResNet-18 has four residual groups, which has channel size of 64, 128, 256, and 512, respectively. Inside each residual group are two standard residual blocks with the same channel size.

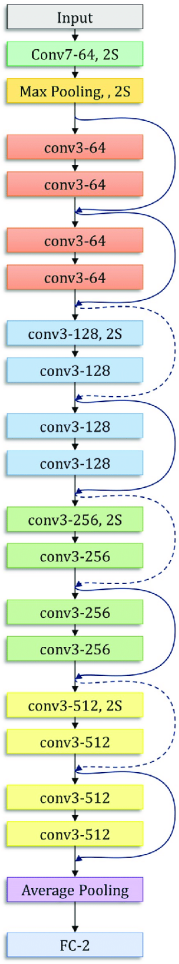

Figure 4.5: An Example of a multi-group ResNet. This ResNet has four residual groups, which has channel size of 64, 128, 256, and 512, respectively.

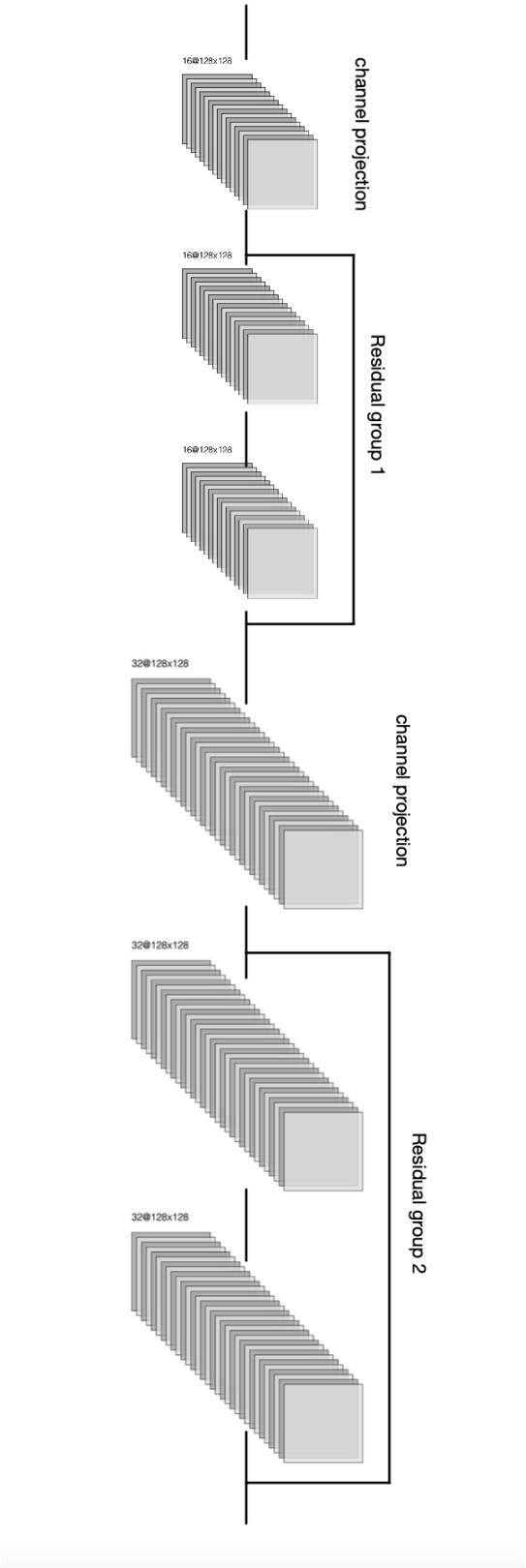

Figure 4.6: An Example of a ResNet suitable for multi-level methods. We need to add a projection layer in front of each residual group.

When we design ResNets suitable for multi-level methods, we need to add a projection layer in front of each residual group. A projection layer includes a two-dimensional CNN layer, batch normalization, and activation function. The role is to ensure that the feature dimension matches the network's CNN kernel. For example, as Figure 4.4 shows, this ResNet has two residual groups. The first residual group has a shape of $(16, 128, 128)$, and the second residual group has a shape of $(32, 128, 128)$. In this case, the projection layer between these two residual groups should be $conv2d(16, 32)$.

The ResNets with multi-group Blocks can be considered as combinations of multiple ODEs. As Figure 4.7 shows, this ResNet contains two different ODEs. When applying multi-level methods to this ResNet, we interpolate from shallow to deep in each residual group using the weight-copied methods.

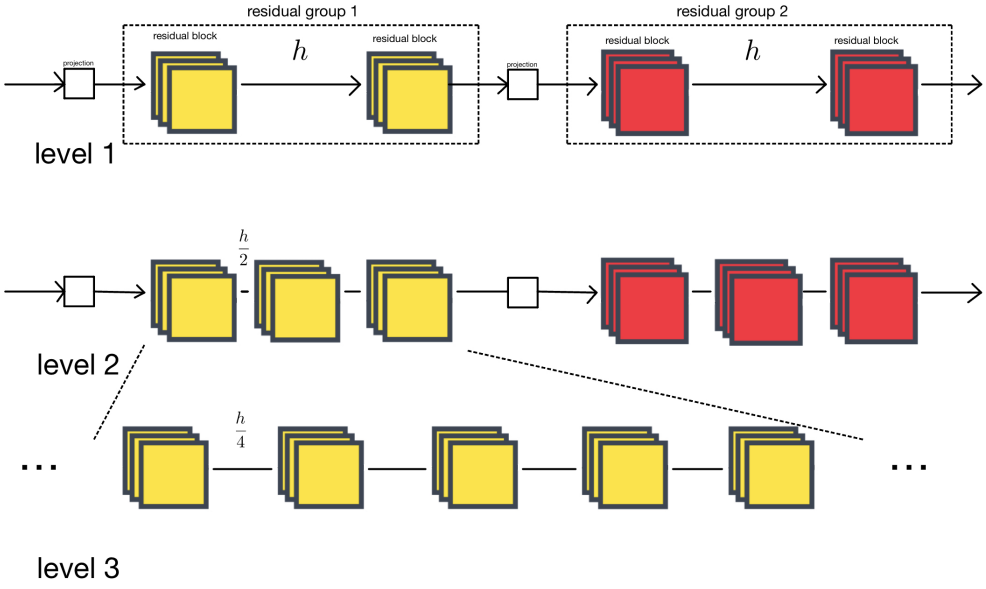

Figure 4.7: An Example of multi-level methods. The ResNets with multi-group Blocks can be considered as combinations of multiple ODEs.

Chapter 5

Experiments of Multi-level Methods for ResNets

In these experiments, we use three datasets to test whether the multi-level methods can reduce the training time of a deep ResNet model while guaranteeing a high level of non-degraded accuracy. The first dataset is an artificial dataset with the regression task, and the other two datasets are image datasets with the classification tasks. For each experiment, we design two sets of baselines: a deep ResNet and a shallow one. We will report the training time and accuracy of the models trained by the multi-level method and compare them to the baselines. Our experimental environment is Google collab with Python 3.10.12 version, PyTorch 2.0.1 version, and CUDA 118 version.

5.1 Multi-level Methods on Artificial Dataset

In this experiment, we generate a complex function consisting of a combination of trigonometric functions as our dataset and compare the result of the model trained by the multilevel method and two different baselines which are two ResNet with different depth.

5.1.1 Artificial Dataset Generation

To conduct our first experiment, we created an artificial dataset based on a specified mathematical function:

$$f(x) = \cos^2(x)\sin(x^2)$$

From Figure 5.1, this function incorporates both square and triangle-metric transformations of the input x, resulting in a complex pattern of oscillations. By choosing such a function, we ensured that our dataset would be non-trivial and sufficiently challenging for our ResNet MLP models.

The dataset was generated by sampling 2000 points uniformly from the interval $[0, 2\pi]$. Each sampled point x was passed through the function $f(x)$ to produce the corresponding target value y.

To make the dataset more realistic and to evaluate the robustness of the models to noise, we introduced Gaussian noise to the target values. The noise was sampled from a normal distribution with a mean of 0 and a standard deviation of $1/500$. This added a small amount of variability to the target values, simulating the kind of noise that is often present in real-world datasets.

To evaluate the performance of our models, we generated a separate test dataset. The test dataset was created using the same mathematical function to ensure consistency with the training data, thus enabling a fair evaluation of the model's ability to generalize to unseen data.

The test dataset was generated by sampling 1,500 points uniformly from the same interval as the training data. This ensures that the test data covers the same input space as the training data but consists of different points, providing a stringent test of the models' generalization capabilities.

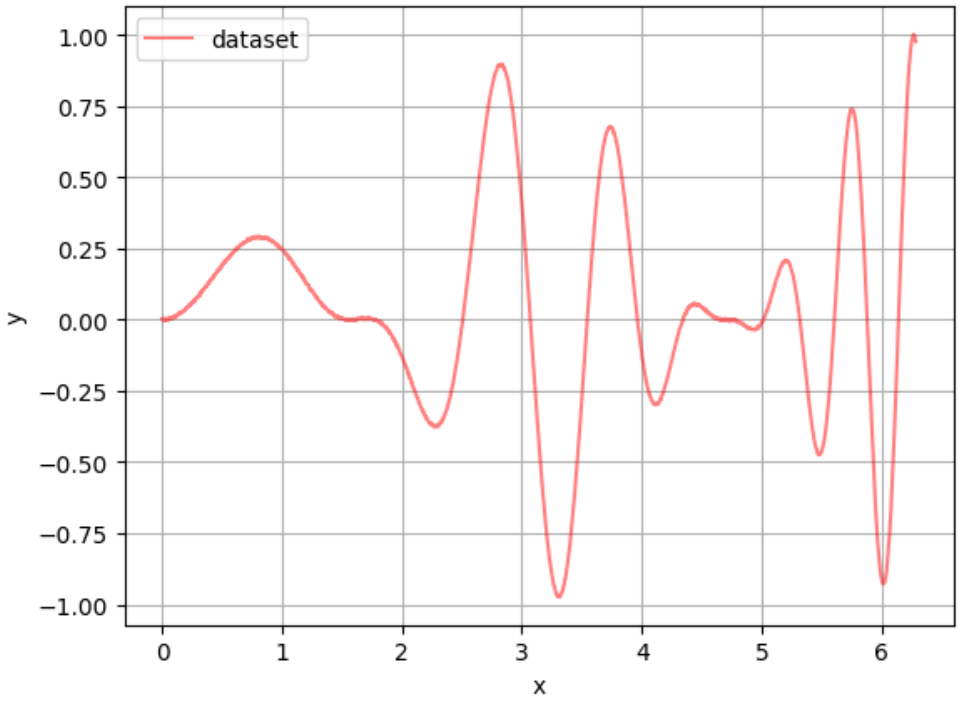

Figure 5.1: The visualisation of this artificial dataset. We add a Gaussian noise to simulate the sampling noice.

5.1.2 Baselines

To comprehensively evaluate our multi-level training method, we established two baseline models representing different degrees of network complexity: a shallow ResNet MLP and a deep ResNet MLP.

From Figure 5.2, the first Baseline model is a shallow ResNet MLP, which has one input layer, one output layer, and two residual blocks as the start and end of an ODE. The input layer maps the input features to the same dimension as the residual blocks, which in the model is 32. The output layer does the opposite of the input layer, mapping the 32-dimensional output of the residual blocks back to the original output space. Each residual block contains a small neural network consisting of two linear transformations (MLP layers). Also, we chose to use the most common activation function (sigmoid).

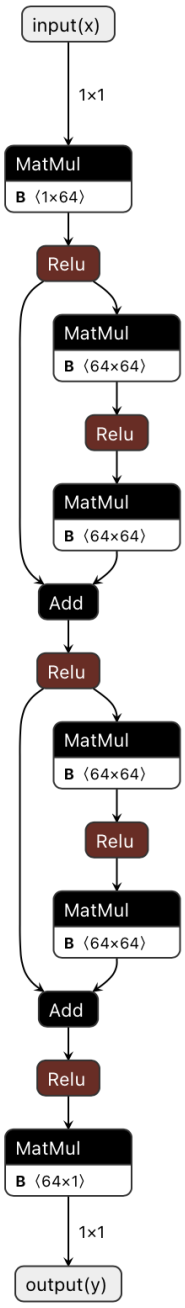

Figure 5.2: The structure of the shallow ResNet MLP. It contains one input layer, one output layer, and two residual blocks. Each residual block contains two MLP layers.

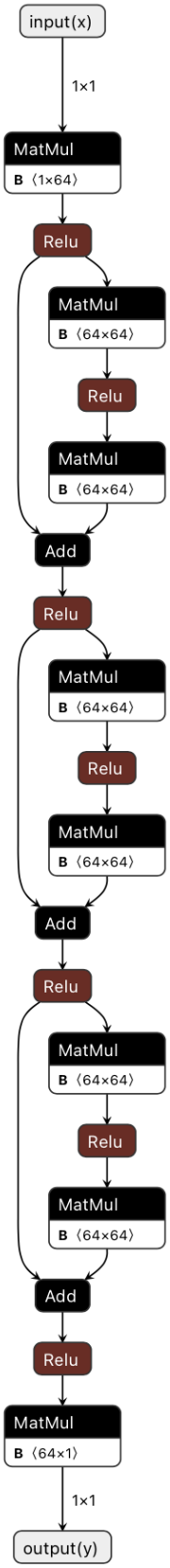

Figure 5.3: The structure of the deep ResNet MLP. It contains three residual blocks.

The second baseline model was a deep ResNet MLP, which was identical to the shallow ResNet MLP but contains three blocks. The increased depth of this model allowed it to learn more complex patterns from the data.

Both models were trained independently on the same training dataset and evaluated on the same test dataset. As for the training of models, we use MSE as the loss function. As for the test of models, we use the second norm of predicted value and ground truth of test dataset. For the training parameters, we choose the learning rate $lr = 0.005$, total epoch of training $n\_epoch = 50$, and batch size $batch\_size = 256$. Here are the experiment results of these two baseline models, as Figure 5.4 and 5.5 shown.

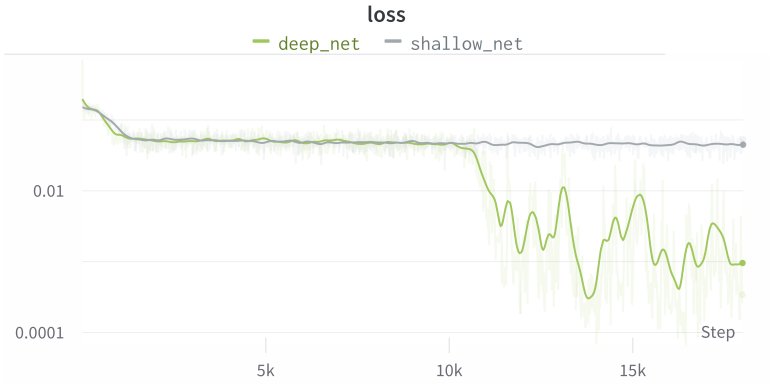

Figure 5.4: The loss function. The *deep_net* means the loss of the deep ResNet MLP, and deep2 means the shallow one.

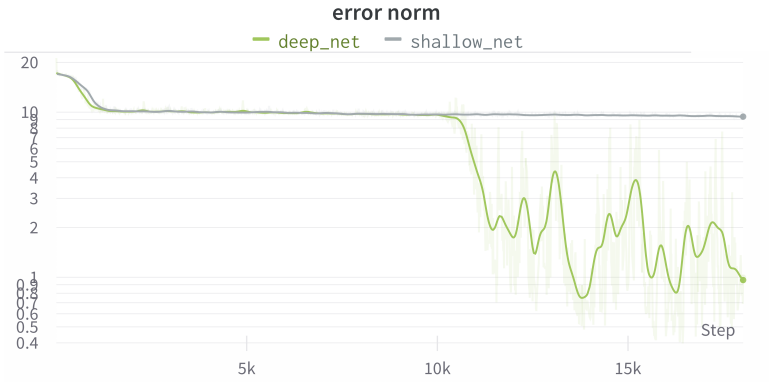

Figure 5.5: The second norm of predicted value and ground truth of test dataset.

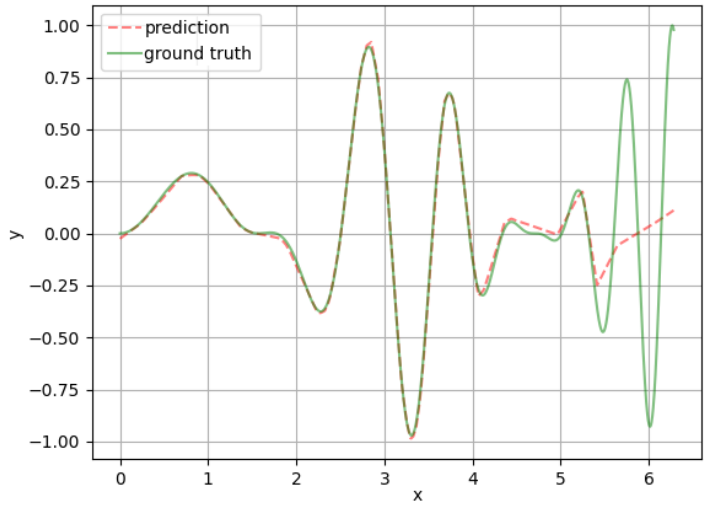

Figure 5.6: The prediction and ground truth of the shallow network. The difference between the predicted and actual values is very large.

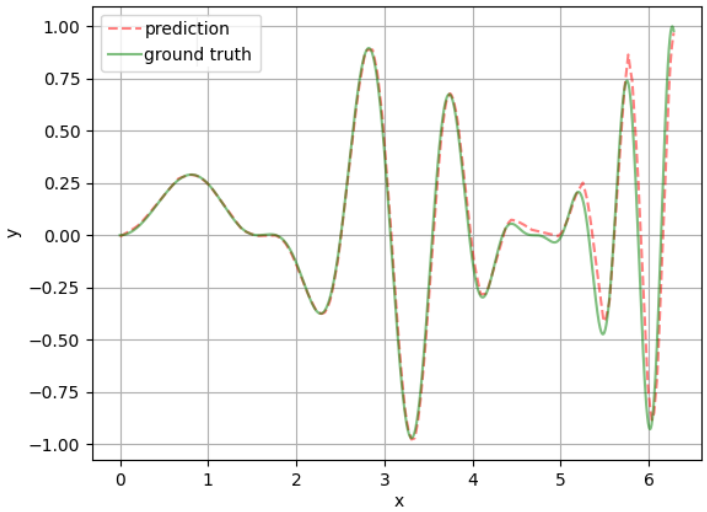

Figure 5.7: The prediction and ground truth of the deep network. The deep network can fit the data properly.

The shallow network cannot learn very complex patterns from data, and there is a limit to this model-fitting ability because there are very few artificial

neurons in the shallower network. From Figure 5.6, we can see that the loss function has converged to about 0.05. However, the difference between the predicted and actual values is huge when $x \in (5.3,\ 2\pi)$. From Figure 5.6, the deep network can fit the data correctly.

5.1.3 Multi-level Methods

The steps of the multi-level training are that we start with a shallow network using a more considerable explicit step h. After a few training epochs, we switch to a deeper network with double the number of residual blocks. Moreover, we halve the step size to $\frac{h}{2}$. Specifically, for our experiment, firstly, the ResNet MLP, as in Fig. 5.2 is used, and the explicit step h is 1. After 25 training epochs, the network is switched to the ResNet MLP as in Figure 5.3, and it changes h to $\frac{h}{2}$. We choose the same parameters as baseline models: learning rate $lr = 0.005$, the total epoch of training $n\_epoch = 50$, and batch size $batch\_size = 256$.

Figure 5.8 shows that the accuracy of the multi-level method is comparable to that of training directly using the deep ResNet MLP. However, we use the shallow network to train the epoch's first half, which means the time complexity is lower than the deep one. In this case, running the first half of the epochs takes less time. From Figure 5.9, the total training time of the multi-level method is 90.3 seconds, which is lower than the time of the deep ResNet MLP (97.8 seconds). And the GPU memory allocation of the multi-level method is same as the deep net.

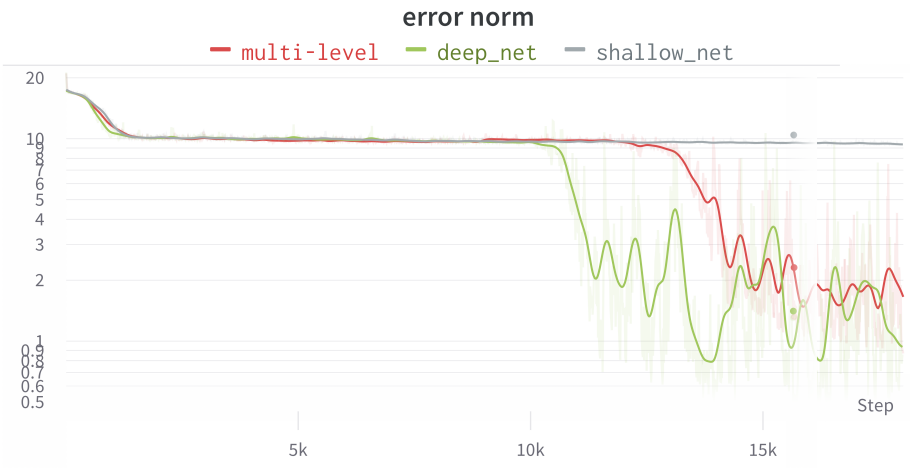

Figure 5.8: The error norm of the test dataset using multi-level methods. The accuracy of the multi-level is same as the one of the deep ResNet MLP.

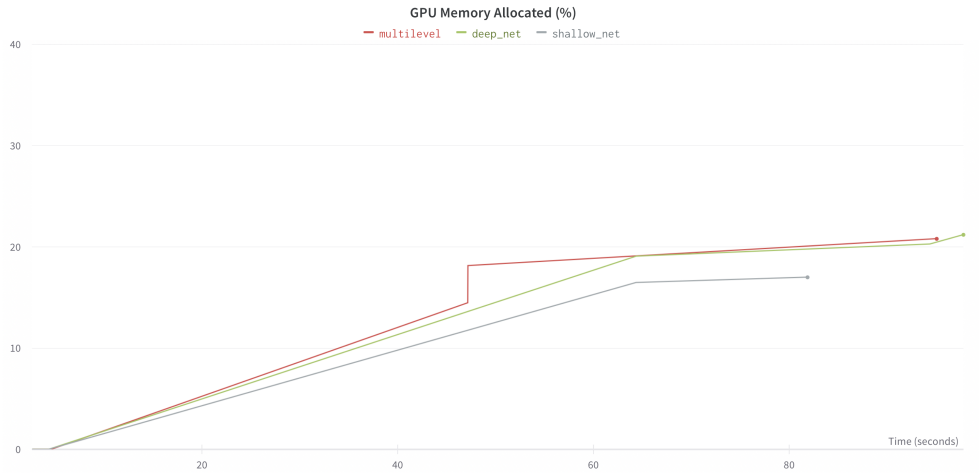

Figure 5.9: The GPU Memory Allocation of the multi-level method is same as the deep net.

5.2 The Multi-level Method on Image Dataset Fashion-MNIST

In this experiment, we explore the application of multi-level methods in computer vision. Image classification is the most classical problem in computer vision. We use Fashion dataset for this experiment and two different baselines which are ResNet12 and ResNet16.

5.2.1 Description of Datasets

The Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. From Figure 5.10, each example is a $(28, 28)$ grayscale image, associated with a label from 10 classes. Because the size of the Fashion-MNIST dataset is small, which makes us possible to conduct this experiment on an average GPU. Also, due to the diversity of categories, the Fashion-MNIST dataset provides a good benchmark to evaluate and compare the performance of the different machine learning algorithms.

5.2.2 Baselines

We established two baseline models representing different degrees of network complexity: a shallow ResNet CNN and a deep ResNet CNN.

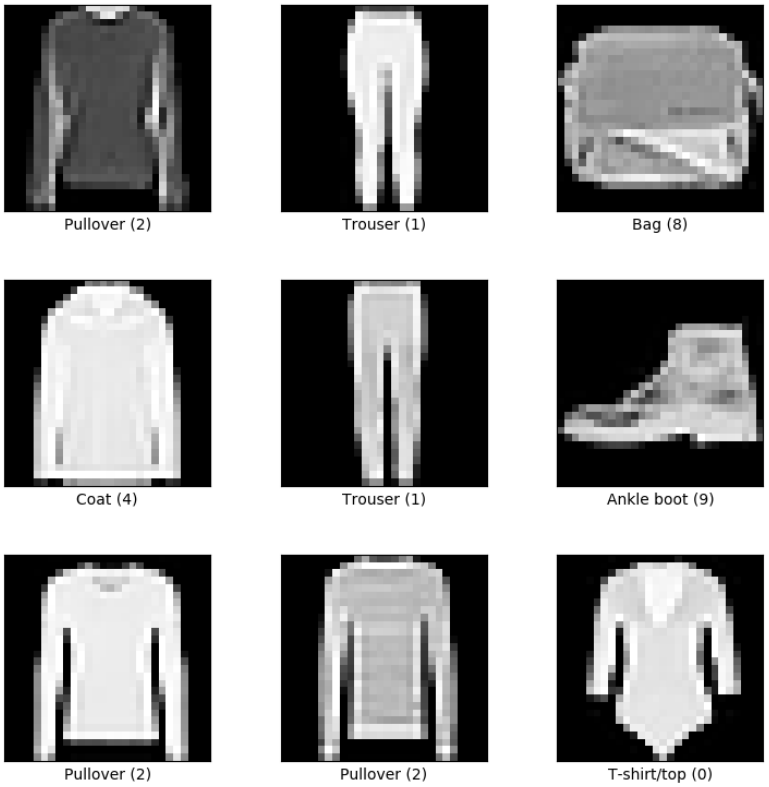

Figure 5.10: Data Visualisation of the Fashion-MNIST dataset. We ramdomly choose 9 pictures from the dataset.

The first baseline model is a shallow ResNet CNN, which has one input layer, one output layer, and two residual groups with their projection layers, as Figure 5.13 shows. The projection layers map the feature shape to the same channel as the residual blocks. There are two residual blocks inside one residual group. Each residual block contains a small neural network, which consists of two CNN layers, batch normalisation layers and actiovation functions. Also, we choose to use the most common activation function sigmoid in my model. The second baseline model was a deep ResNet CNN, which contains three residual blocks inside each residual groups.

Both models were trained independently on the same training dataset and evaluated on the same test dataset. As for the training of models, we use cross entropy as the loss function. For the training parameters, we choose

the learning rate $lr = 0.005$, total epoch of training $n\_epoch = 50$, and batch size $batch\_size = 1024$. Here are the experiment results of these two baseline models, as Figure 5.11 and 5.12 shown. We can observe that on the test dataset, the deep ResNet has a 91.14% accuracy better than shallow one 90.44%.

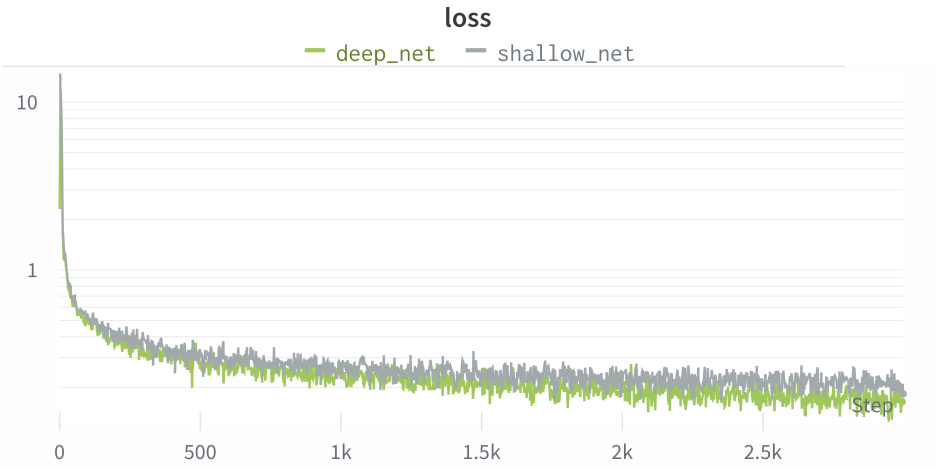

Figure 5.11: The loss function of two baselines. The deep CNN's loss function is slightly lower than the shallow one.

Figure 5.12: The accuracy of two baselines in test dataset. The deep ResNet has a higher accuracy.

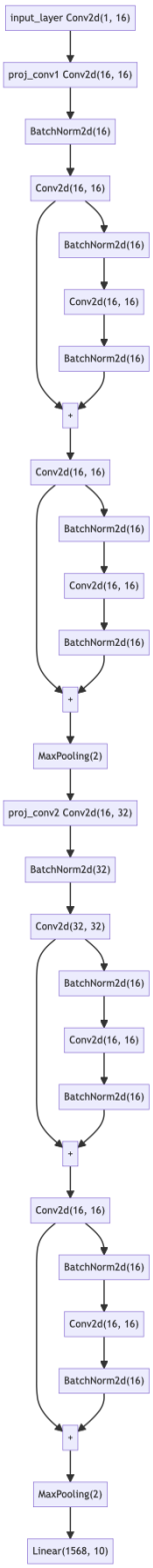

Figure 5.13: The structure of the shallow ResNet. There are two residual groups with their projection layers inside this ResNet. The second baseline contains three residual blocks inside each residual groups.

5.2.3 Multi-level Methods

We start with the shallow ResNet using a larger explicit step $h = 1$. After 25 training epochs, we switch to the deeper ResNet. And we halve the step size to $\frac{1}{2}$.

From Figure 5.14 shown, the accuracy of the multi-level method (91.56%)is almost same with training directly using the deep ResNet. However, it takes less time to run the first half of the epochs. From Figure 5.16, the totally training time of the multi-level method is 513.46 seconds, which is lower than the time of the deep ResNet (539.72 seconds).

While switching the network from shallow to deep, we find an unsmooth jump in the loss function from Figure 5.14. Because the residual networks are the numerical computations of the ODE systems. When switching networks, the loss value of the deeper network is not exactly equal to the loss of the shallower network due to the error in the numerical calculation. The batch size mainly decides the GPU memory allocations of image datasets during model training. Because the GPU memories of the neural networks are significantly smaller than the image dataset. Thus, from Figure 5.16, the GPU memories of the multi-level method is approxiamtely same as two baselines.

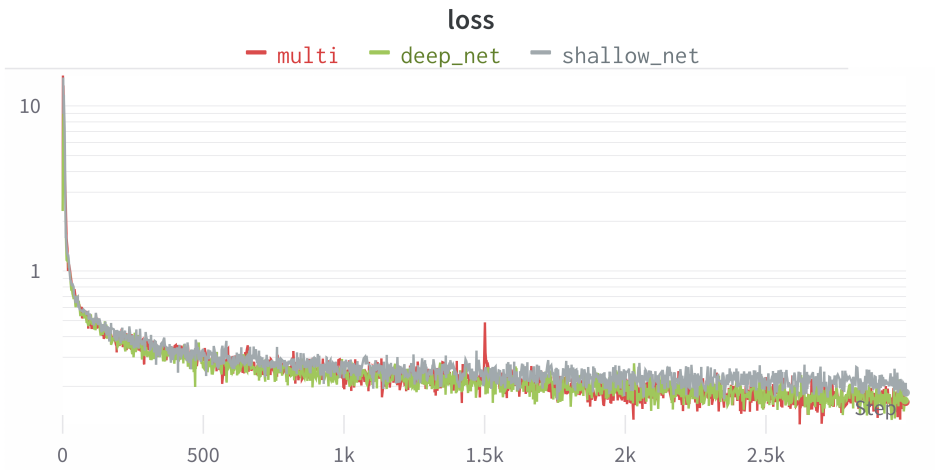

Figure 5.14: The loss function of the multi-level method. There is a jump when changing to deep network.

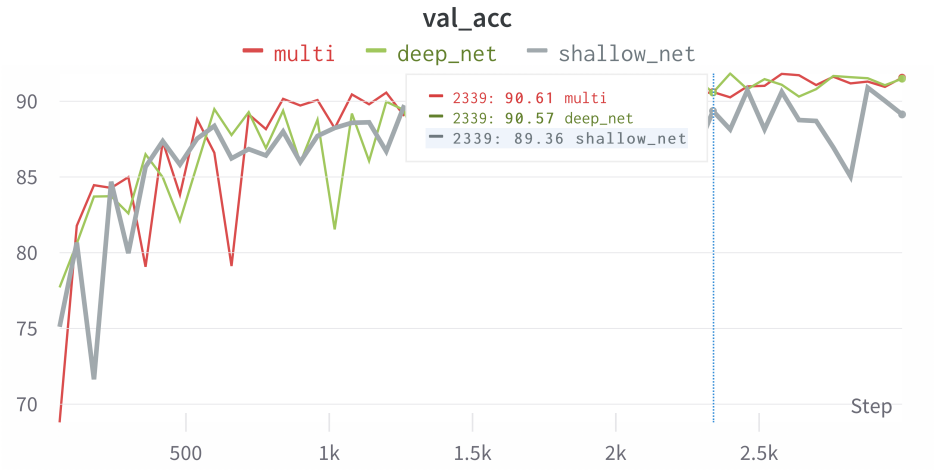

Figure 5.15: The accuracy of the multi-level method in test dataset. the accuracy of the multi-level method is almost same with the deep ResNet

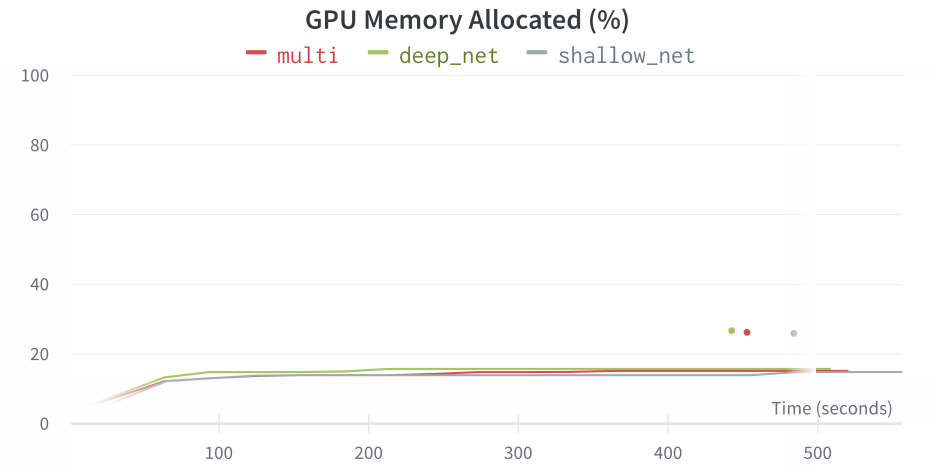

Figure 5.16: The GPU memories of the multi-level method is approxiamtely same as two baselines.

5.3 The Multi-level Method on Image Dataset CIFAR-10

In this experiment, we explore the application of multi-level methods in computer vision. Image classification is the most classical problem in computer vision. We use CIFAR-10 as the dataset for this experiment and two different baselines which are ResNet-12 and ResNet-24.

5.3.1 Description of Datasets

The CIFAR-10 is a small image dataset for the image classification task. From Figure 5.17, each data in CIFAR-10 is a $(32, 32)$ image with three channels for the color expression, and there is a label for each image [21]. The categories of labels include: airplane, car, bird, cat, deer, dog, frog, horse, boat and truck. The CIFAR-10 dataset is divided into 50,000 training images and 10,000 test images.

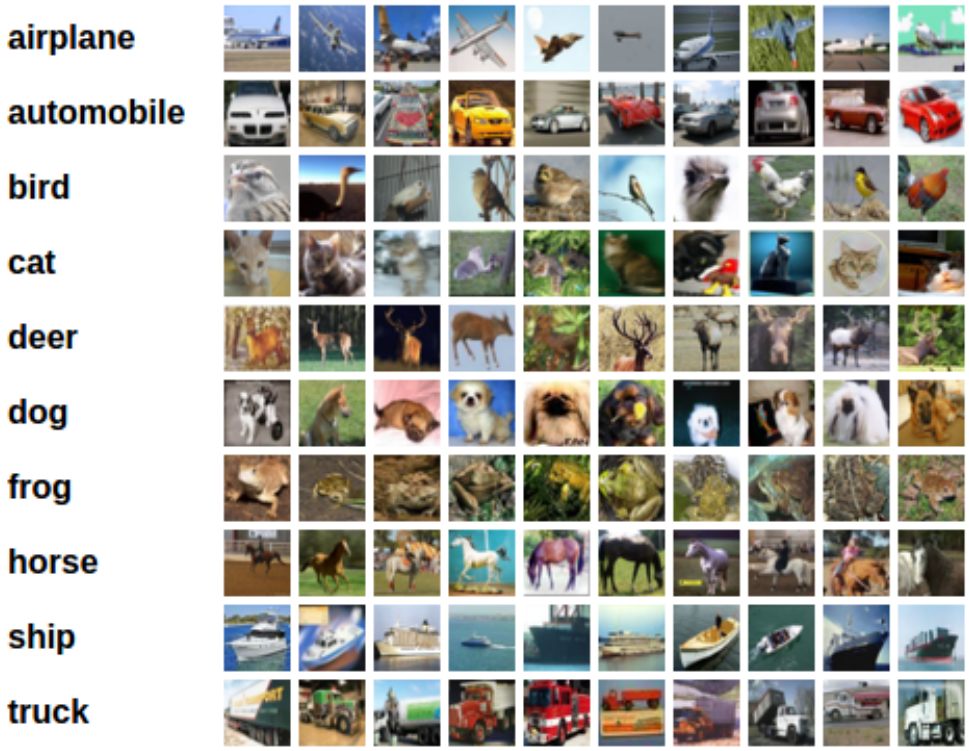

Figure 5.17: Data Visualisation of the CIFAR-10 dataset. Each data in CIFAR-10 is a $(32, 32)$ image with three channels for the color expression.

5.3.2 Baselines

We established two baseline models representing different degrees of network complexity: a shallow ResNet CNN and a deep ResNet CNN.

From Figure 5.18, the first Baseline model is a shallow ResNet CNN, with one input layer, one output layer, and three residual groups with their projection layers. There are two residual blocks inside one residual group. Each residual block contains a small neural network consisting of two CNN layers, batch normalization layers, and activation functions. The second baseline model was a deep ResNet CNN, containing five residual blocks inside each group.

Both models were trained independently on the same training dataset and evaluated on the same test dataset. As for the training of models, we use cross entropy as the loss function. We choose the learning rate $lr = 0.005$, the total epoch of training $n\_epoch = 50$, and batch size $batch\_size = 1024$ for the training parameters. From Figure 5.20, we can observe that the loss of the deep ResNet is lower than the shallow one. From Figure 5.18, the deep ResNet has a 90.18% accuracy higher than the shallow one 87.26% on the test dataset.

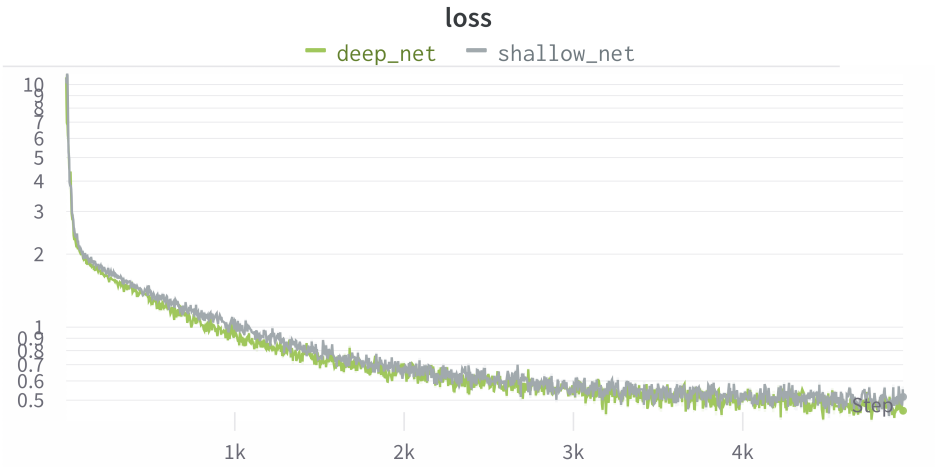

Figure 5.18: The plot of the loss function of two baselines. The loss of the deep ResNet is lower than the shallow one.

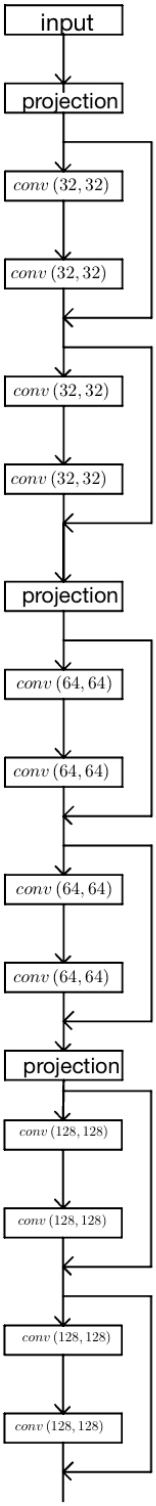

Figure 5.19: The first Baseline model contains two residual blocks inside one residual group.

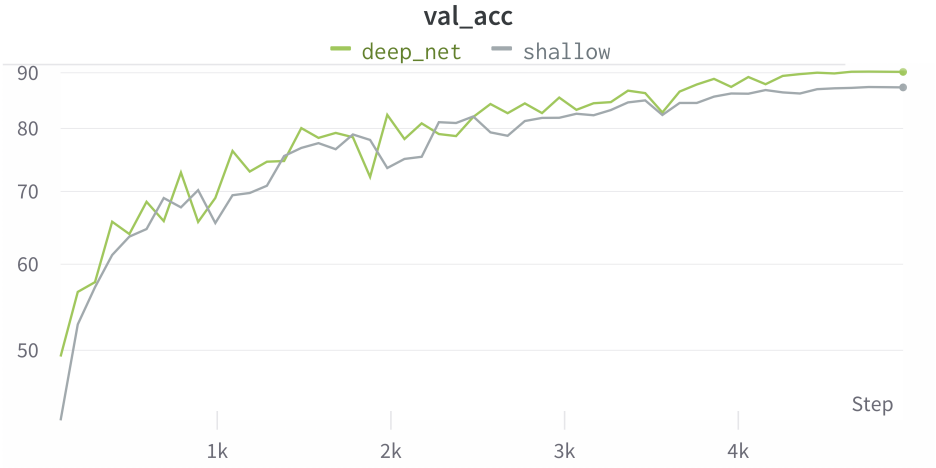

Figure 5.20: The plot of the accuracy of two baselines. The accuracy of the deep ResNet is higher than the shallow one.

5.3.3 Multi-level Methods

In this experiment, we used three-level ResNets for multi-level methods, as Figure 5.22 shows. We start with the shallow ResNet using a larger explicit step $h = 1$. After a 15 training epochs, we switch to a deeper ResNet with three residual blocks inside one residual group. And we halve the step size to $\frac{1}{2}$. Finally, after another 15 training epochs, we switch to the deep ResNet baseline with five residual blocks inside one residual group.

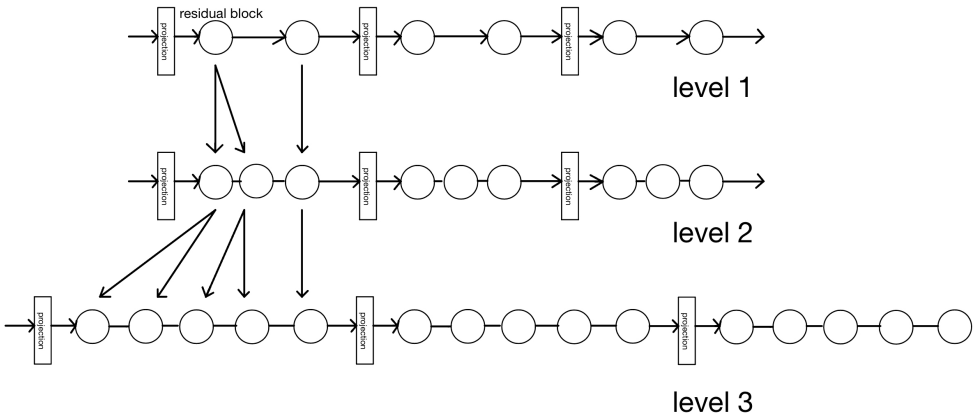

Figure 5.21: The three-level ResNets for multi-level methods.

From Figure 5.23, the accuracy of the multi-level method (90.51%)is almost same with training directly using the deep baseline (90.18%), and much

better than the shallow baseline (87.26%). However, the training time of the multi-level methods (23.68 minutes) is significantly lower than the deep baseline (30.14 minutes).

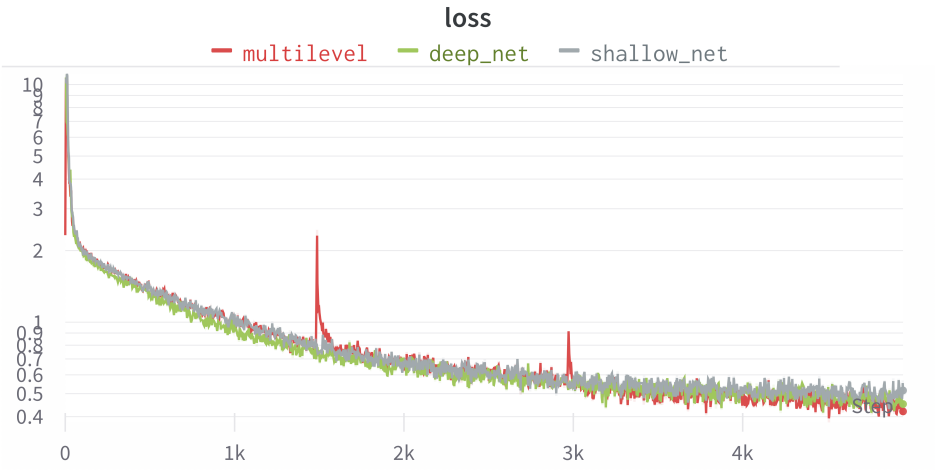

Figure 5.22: The loss function of the multi-level methods.

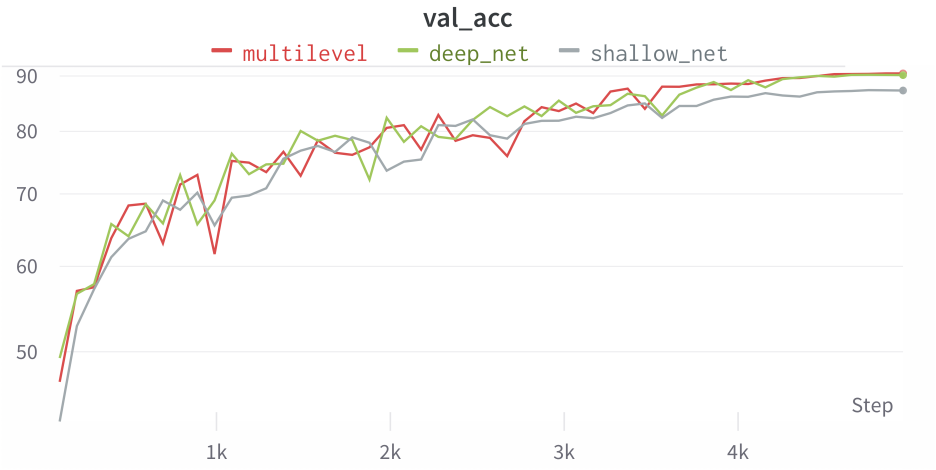

Figure 5.23: The three-level ResNets for multi-level methods. These arrows mean the weight copies.

Chapter 6

Experiments of ODEs for ResNets

In these experiments, we use two dataset to test whether the RK methods can increase the model accuracies. The first dataset is an artificial dataset with the regression task, and another two datasets are image datasets with the classification tasks. Finally, we will test a new method which is the combination of RK methods and multi-level methods. This new method can not only increase the model accuracies for regression tasks, but also decrease the training time. Our experimental environment is google colab with Python 3.10.12 version, PyTorch 2.0.1 version and CUDA 118 version.

6.1 The RK's Method on The Artificial Dataset

6.1.1 Dataset and Baseline

From Chaper 5.1.1, we use same dataset with same evaluation methods. From Chapter 5.1.2 and Figure 5.2, we use the ResNet MLP with 2 residual blocks as the baseline.

6.1.2 The RK method

From Chapter 3.6 and Figure 5.2, we choose the same number of the residual blocks. As Figure 6.1 shown, we modify each block to the residual block with the RK2 method. Then, we obtain a ResNet with the RK2 method. We choose same parameters as baseline models: learning rate $lr = 0.005$, total epoch of training $n\_epoch = 50$, $h = 1$ and batch size $batch\_size = 256$.

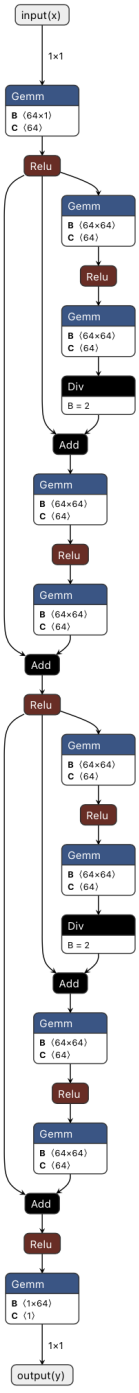

Figure 6.1:

From Figure 6.2 shown, the performance of ResNet with the RK2 method is higher than the baseline ResNet MLP. Because the error norm of the RK2

method (0.439) in test dataset is lower than the baseline (9.792). However, the time complexity of the RK2 method is lower. In this case, it takes more time for model training. From Figure 6.4, the totally training time of the RK2 method is 88.4 seconds, which is higher than the time of the baseline (81.8 seconds). And the GPU memory allocation of the ResNet with the RK2 method is same as the baseline, which shows our theory (Chasper 3.7) is correct.

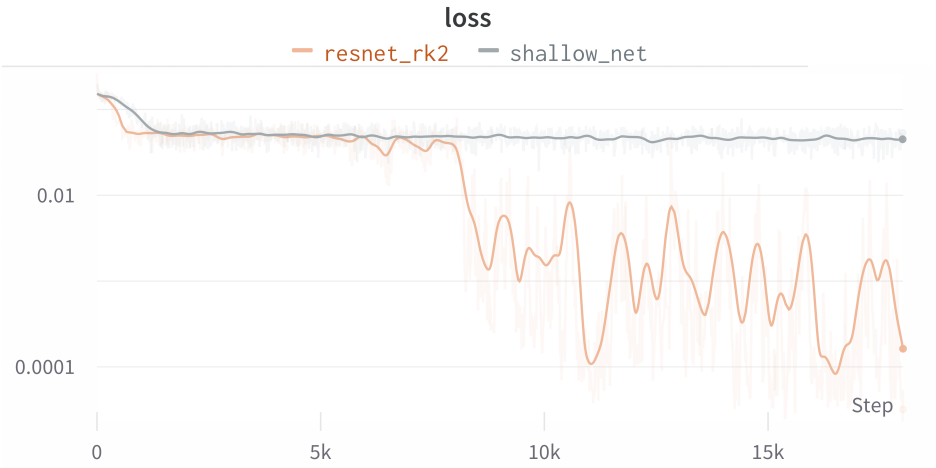

Figure 6.2: The error norm of the RK2 method in test dataset is lower than the baseline.

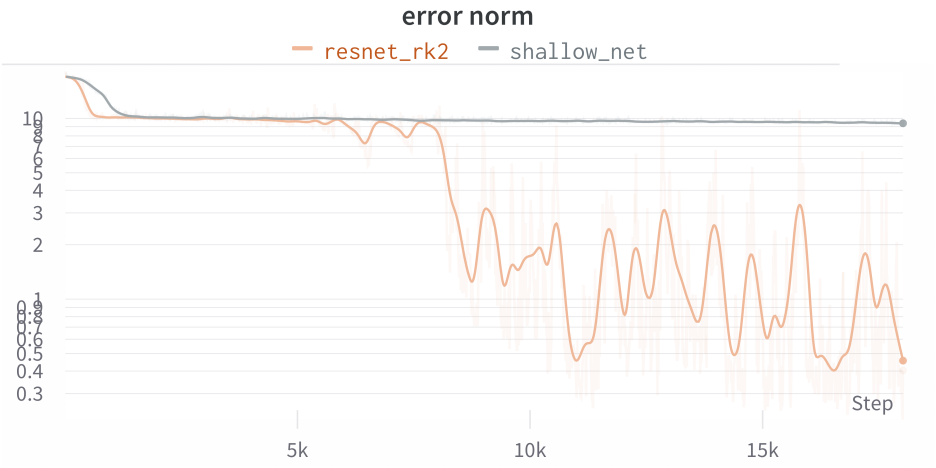

Figure 6.3: The training time of the RK2 method is higher than the time of the baseline.

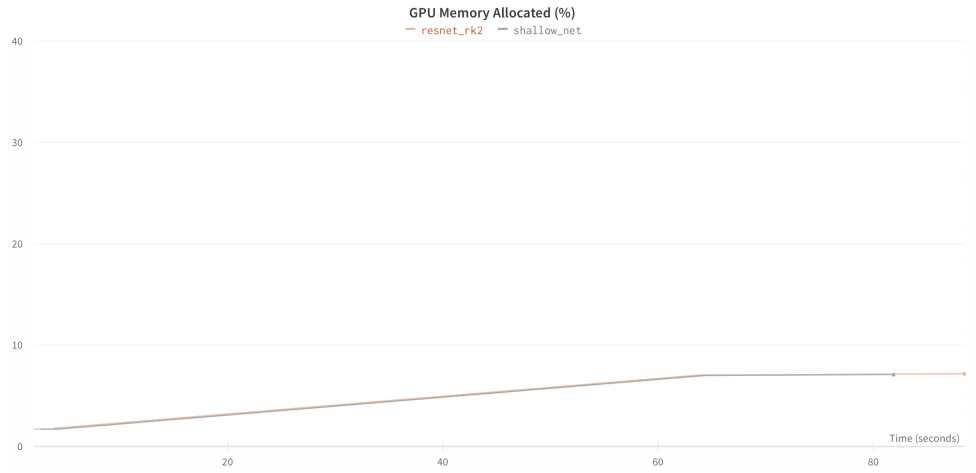

Figure 6.4: The GPU memory allocation of the ResNet with the RK2 method is same as the baseline.

6.2 The RK's Method on The Fashion Dataset

6.2.1 Dataset and Baseline

From Chaper 5.2.1, we use same image dataset. From Chapter 5.2.2 and Figure 5.11, we use the ResNet CNN with 2 residual groups as the baseline.

6.2.2 The RK's method

We choose the same number of the residual blocks from the baseline model, and modify each block to the residual block with the RK's method. For the training parameters, we choose the learning rate $lr = 0.005$, total epoch of training $n\_epoch = 50$, and batch size $batch\_size = 1024$.

From Figure 6.5 shown, the accuracy of the RK2 method (91.5%) is almost same with the accuracy of the baseline ResNet CNN (91.47%).

Figure 6.5: The accuracy of the RK2 method is almost same with the accuracy of the baseline.

The reason is that most pairs of points are approximately equidistant in a high-dimensional space, as Figure 6.6 shows. This phenomenon is a manifestation of the curse of dimensionality. From Figure 6.7, assume that the domain of all data is a cube (hypercube) with volume 1. The volume of the inner ball (hypersphere) of this cube (hypercube) is:

$$V(d) = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} 0.5^d,$$

where d is the dimension of the cube (hypercube). When the dimension increases, most of the training data is distributed at the corners of the hypercube, and the distances between each data increase.

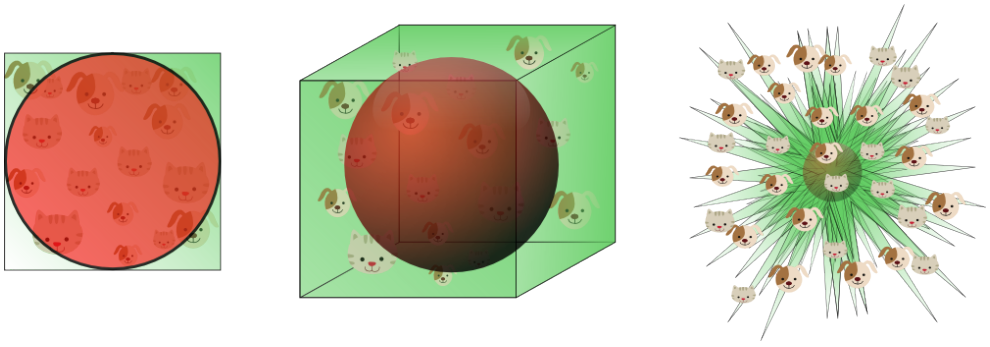

Figure 6.6: When the dimension increases, most of the training data is distributed at the corners.

When training a deep learning model with many parameters, the relative

distances between different classes or clusters in the feature space can become quite large from Figure 6.6, which means that the decision boundaries can be well-separated in classification tasks.

In this case, given the properties above of high-dimensional spaces, the exact precision of the loss value becomes less critical. Because the decision boundaries are already well-separated, small perturbations in the loss value are less likely to change the overall decision of the model. Thus, reducing precision like $fp16$ and $int8$ might not significantly affect the model's ability to discern between different classes. For the same reason, the ResNet with RK2 methods does not affect the accuracy improvement of the classification problems.

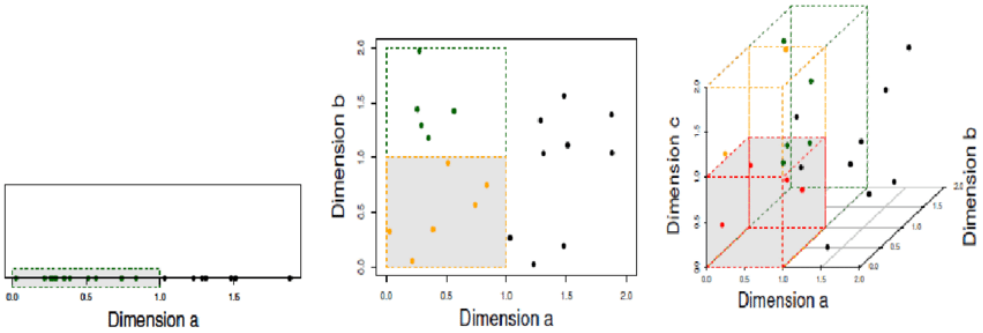

Figure 6.7: The relative distances between different classes or clusters in the feature space can become quite large. Because deep learning models has high dimensions [22].

6.3 Combination of RK2 and Multi-level Methods

6.3.1 Dataset and Baseline

From Chapter 5.1.1, we use same data generation methods. However, we generate a new dataset of a more complex function than the one from Chapter 5.1.1, which is:

$$f(x) = \cos(x^2)\cos(x)\sin(x^2).$$

From Chapter 6.1.2 and Figure 6.1, we use the ResNet MLP with RK method as the baseline, but the number of residual blocks in this experiment is five.

6.3.2 Combination

Figure 6.8 shows the three-level model interpolations of the multi-level methods. The first multi-level method is to start with the level-1 model with the explicit step $h = 1$. The level-1 model is a simple ResNet MLPs, the same

as Figure 5.2. After a 1000 training epoch, we switch to the level-2 model with three residual blocks. Moreover, we halve the step size to $\frac{1}{2}$. After another 3000 training epochs, we switch to the level-3 model and halve the step size to $\frac{1}{4}$.

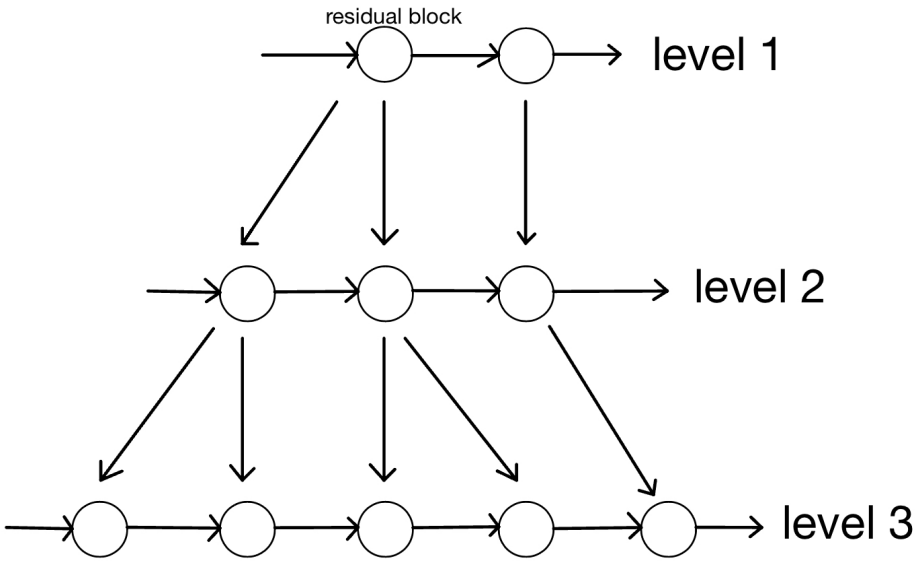

Figure 6.8: The multi-level method with a three-level structure.

From Chapter 3.6, the number of neurons and parameters in our new-designed multi-level methods with RK methods is consistent with the original multi-level methods.

The second multi-level method has the same process of model interpolations. However, we use the ResNet MLPs with the RK method to design models of level two and level three. In the first interpolation, we increase the residual blocks and change the model structure from simple ResNet MLPs to the ResNet MLPs with the RK method. We call this multi-level method a combination of RK2 and multi-level methods.

As for the training of models, we choose the learning rate $lr = 0.001$, the total epoch of training $n\_epoch = 8000$, and batch size $batch\_size = 512$ for the training parameters. Figures 6.9 and 6.10 show that the loss function and error norm of the combination of RK2 and multi-level methods are all lower than the simple multi-level methods. From Figure 6.11, the GPU memory allocation of the multi-level with the RK2 method is same as the standard multi-level method.

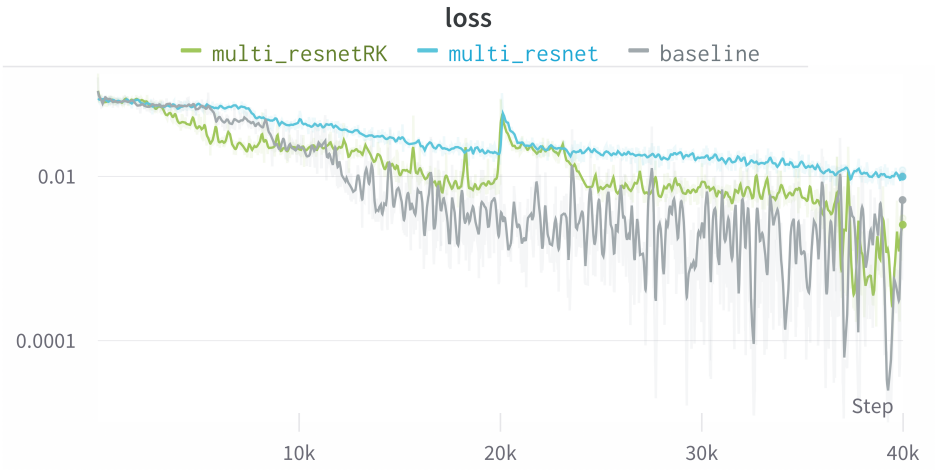

Figure 6.9: The loss function. The baseline is the ResNet MLP with RK method and five residual blocks. The models in the *multi_resnet* is the simple ResNet MLPs. And the models in the *multi_resnetRK* is the ResNet MLPs with RK methods.

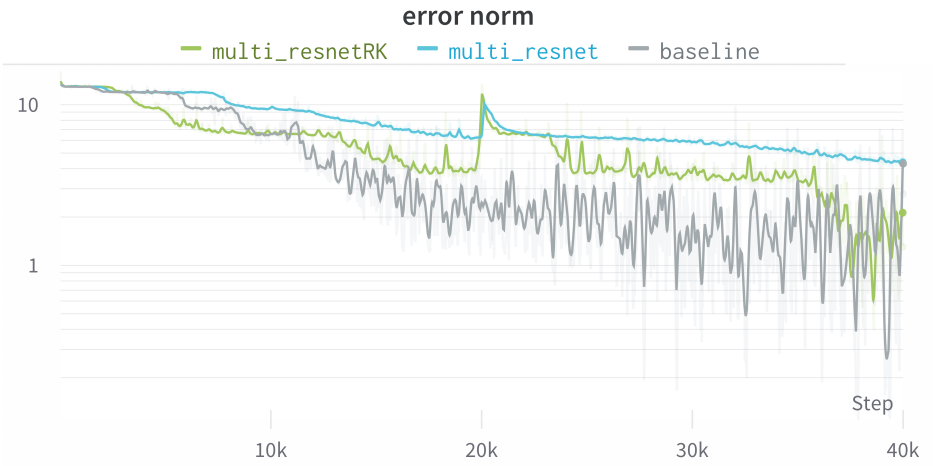

Figure 6.10: The error norm of the combination of RK2 and multi-level methods is lower than the simple multi-level methods.

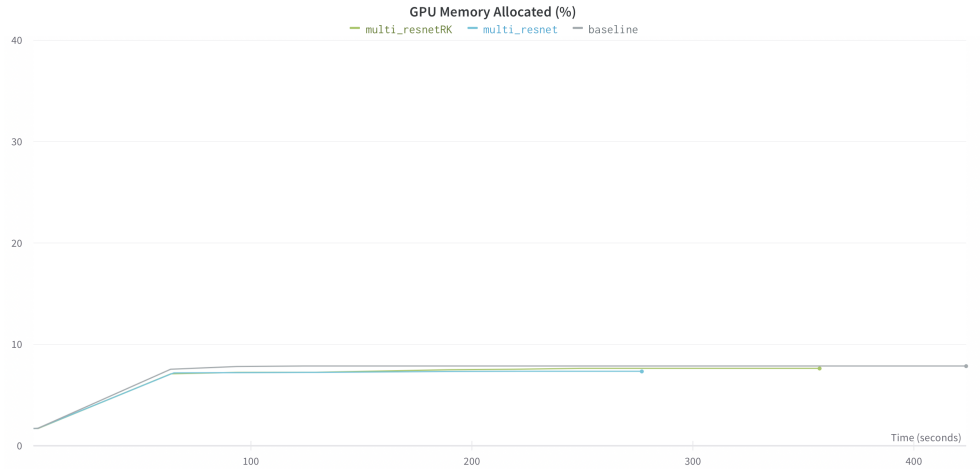

Figure 6.11:

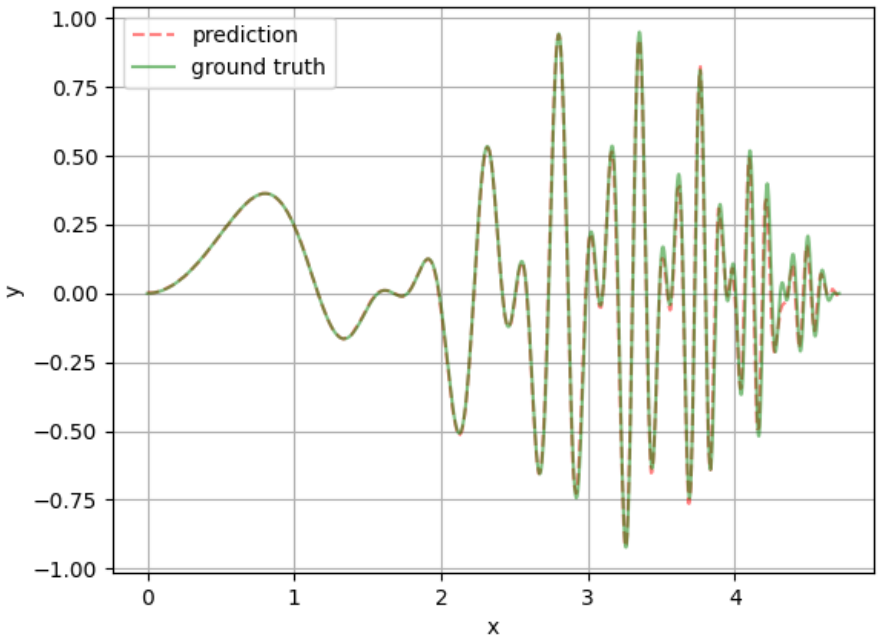

Figure 6.12: The visualization of the combination method, which can simulate complex function.

As we can see from the Figure 6.12 and 6.13, the simple multi-level methods are completely unable to fit the final part of the complex function. However, the combination method, which has been modified by us, is able to simulate

it very well. The training time of the multi-level methods with RK2 methods (357.33 seconds) is higher than the standard multi-level methods (279.31 seconds).

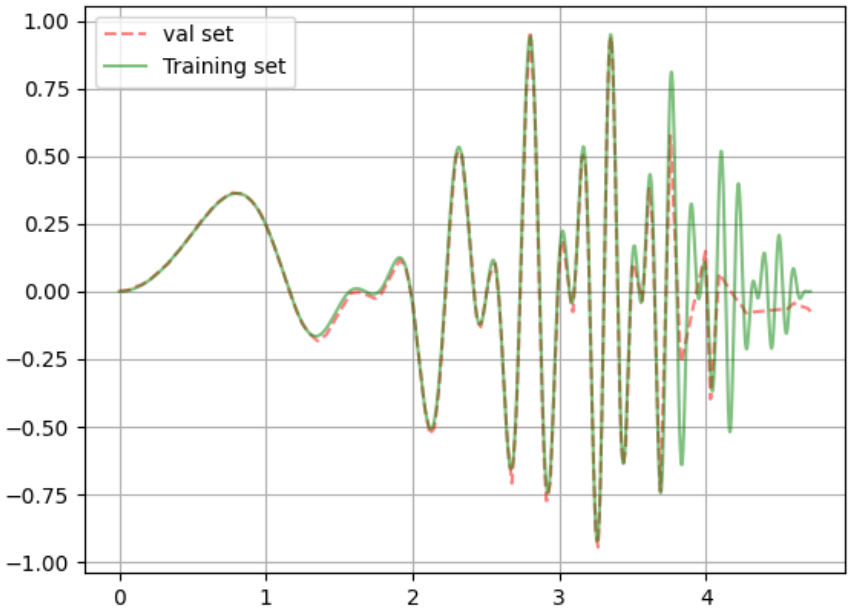

Figure 6.13: The visualization of the simple method, which cannot simulate the final part of the complex function.

Chapter 7

Conclusion

This thesis explores multilevel methods and numerical computation of ODEs in residual networks. The multilevel approach can be applied to the training of deep residual networks. This method has significant computational advantages. The ODE numerical computation method can be applied to improve the model's accuracy in regression problems. We combine these two methods to create residual networks for the multilevel training RK method. Our combined algorithm not only speeds up the training process of ResNets but also ensures that the model accuracy is not compromised. It combines the interpretability of dynamic system models with the accuracy required for deep learning, setting a new benchmark for ResNet training methods. This work could be applied to future 3D reconstruction tasks in computer vision. Since this human-free belongs to the regression problem in computer vision, it fits our approach perfectly.

Bibliography

[1] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[2] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

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

[4] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[5] Gustavo Albuquerque Lima. Ways to Improve Your Deep Learning Model - Batch Normalization and Adam. https://www.linkedin.com/pulse/ways-improve-your-deep-learning-model-batch-adam-albuquerque-lima/?trackingId=CE2022. Accessed: April 20, 2023.

[6] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[7] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[8] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *J. Open Source Softw.*, 4(33):747, 2019.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 2016.

[10] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. *arXiv preprint arXiv:1710.10348*, 2017.

[11] Fengxiang He, Tongliang Liu, and Dacheng Tao. Why resnet works? residuals generalize. *IEEE transactions on neural networks and learning systems*, 31(12):5349–5362, 2020.

[12] Neili Zakaria, Fezari Mohamed, Redjati Abdelghani, and Kenneth Sundaraj. Three resnet deep learning architectures applied in pulmonary pathologies classification. pages 1–8, 2021.

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

[14] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[15] Claudio Filipi Gonçalves Dos Santos and João Paulo Papa. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Computing Surveys (CSUR)*, 54(10s):1–25, 2022.

[16] Howard Howie Weiss. The sir model and the foundations of public health. *Materials matematics*, pages 0001–17, 2013.

[17] Serge Gratton, Alena Kopanicáková, and Ph L Toint. Multilevel objective-function-free optimization with an application to neural networks training. *arXiv preprint arXiv:2302.07049*, 2023.

[18] Alena Kopaničáková. On the use of hybrid coarse-level models in multilevel minimization methods. *arXiv preprint arXiv:2211.15078*, 2022.

[19] Manel Soria Guerrero. Introduction to the numerical solution of the navier-stokes equations. module 13: Boundary conditions. the driven cavity problem. 2021.

[20] Taras Gerya. *The multigrid method*, page 292–318. Cambridge University Press, 2 edition, 2019.

[21] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[22] Dharmveer Singh Rajput, Pramod Kumar Singh, and Mahua Bhattacharya. Iqram: a high dimensional data clustering technique. *International Journal of Knowledge Engineering and Data Mining*, 2(2-3):117–136, 2012.

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.