이학박사 학위논문

Universal Approximation in Deep Learning

딥러닝에서의 보편 근사 정리

2023년 2월

서울대학교 대학원

수 리 과 학 부

황 건 호

Universal Approximation in Deep Learning

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Geonho Hwang

Dissertation Director : Professor Myungjoo Kang

Department of Mathematical Sciences
Seoul National University

February 2023

Universal Approximation in Deep Learning

딥러닝에서의 보편 근사 정리

지도교수 강 명 주

이 논문을 이학박사 학위논문으로 제출함

2022년 10월

서울대학교 대학원

수 리 과 학 부

황 건 호

황 건 호의 이학박사 학위논문을 인준함

2022년 12월

위 원 장 ________________________ (인)

부위원장 ________________________ (인)

위 원 ________________________ (인)

위 원 ________________________ (인)

위 원 ________________________ (인)

Abstract

Universal Approximation in Deep Learning

Geonho Hwang

Department of Mathematical Sciences

The Graduate School

Seoul National University

Universal approximation, whether a set of functions can approximate an arbitrary function in a specific function space, has been actively studied in recent years owing to the significant development of neural networks. Neural networks have various constraints according to the structures, and the range of functions that can be approximated varies depending on the structure. In this thesis, we demonstrate the universal approximation theorem for two different deep learning network structures: convolutional neural networks and recurrent neural networks.

First, we proved the universality of convolutional neural networks. A convolution with padding outputs the data of the same shape as the input data; therefore, it is necessary to prove whether a convolutional neural network composed of convolutions can approximate such a function. We have shown that convolutional neural networks can approximate continuous functions whose input and output values have the same shape. In addition, the minimum depth of the neural network required for approximation was presented, and we proved that it is the optimal value. We

also verified that convolutional neural networks with sufficiently deep layers have universality when the number of channels is limited.

Second, we investigated the universality of recurrent neural networks. A recurrent neural network is past dependent, and we studied the universality of recurrent neural networks in the past-dependent function space. Specifically, we demonstrated that a multilayer recurrent neural network with limited channels could approximate arbitrary past-dependent continuous functions and L_p functions, respectively. We also extended this result to bidirectional recurrent neural networks, GRU, and LSTM.

Key words: Universal approximation, Recurrent Neural Network, Convolutional Neural Network, Deep Narrow Network

Student Number: 2017-25155

Contents

Chapter 1

Introduction

Deep learning, a type of machine learning that approximates a target function using a numerical model called an artificial neural network, has shown tremendous success in diverse fields, such as regression [12], image processing [7, 46], speech recognition [1], and natural language processing [24, 23]. While the excellent performance of deep learning is attributed to a combination of various factors, it is reasonable to speculate that its notable success is partially based on the *universal approximation theorem*, which states that neural networks are capable of arbitrarily accurate approximations of the target function. Formally, for any given function f in a target class and $\epsilon > 0$, there exists a network $\mathcal{N}$ such that $\|f - \mathcal{N}\| < \epsilon$. In topological language, the theorem states that a set of networks is dense in the class of the target function. In this sense, the closure of the network defines the range of functions it network can represent.

As universality provides the expressive power of a network structure, studies on the universal approximation theorem have attracted increasing attention. Examples include the universality of multi-layer perceptrons (MLPs), the most basic neural networks [6, 18], and the universality of recurrent neural networks (RNNs) with the target class of open dynamical systems [36]. Recently, in [48], the authors has demonstrated the universality of convolutional neural networks.

However, in some fields, the universal property is barely studied due to its complex structure. Deep recurrent neural networks and convolutional neural networks are two representative examples. In the case of the convolutional neural network, convolution makes the complicated relationship between each component of the function represented by a convolutional neural network. In Chapter 2, we studied the universal property of the convolutional neural network as a function from sequence to sequence. We scrutinize the translation equivariance induced by the idealized convolution neural network without padding and the asymmetry induced by zero padding and its correlation.

The deep recurrent neural network also has similar complexity, too. The network propagates the data through time direction and depth direction, which makes the grid and creates complex interactions at the points of the grid. In Chapter 3, we investigated the universal property of the deep narrow recurrent neural network. We combinatorially analyze the linear deep narrow recurrent neural network and utilize the result to get the universal property of general deep narrow recurrent networks.

1.1 Convolutional Neural Network

The convolutional neural network(CNN) [32, 25], one of the most widely used deep learning modules, has achieved tremendous accomplishment in numerous fields, including object detection [45], image classification [10], and sound processing [40].

Starting with the most basic architecture, LeNet5 [25], many well-known deep learning models such as VGGNet [37], ResNet [16], and ResNeXt [42] have been constructed based on CNN. In this regard, it would be natural to be interested in the universal property of CNN, which justifies using a specific network.

However, despite its extensive range of applications, research on the universal property of CNN has been barely conducted. One of the rare studies is [48]. The paper considered the convolutional neural network with a linear layer combined in the last layer and proved the universal property of the network as the function from $\mathbb{R}^d$ to $\mathbb{R}$. However, networks sometimes are expected to retain the output data in the same shape as the input data. Representative examples include object segmentation [28], depth estimation [4], or image processing such as deblurring [47], inpainting [39], and denoising [11]. Another common usage of CNN is as a feature extractor. The feature extractor extracts information from the data and feeds it to the latter part of the deep learning model. Typically, the feature extracted by CNN is multi-dimensional, and to achieve the purpose of being a module that can be used in common across multiple networks, CNN needs to have the universal property. Also, the paper assumed an unrealistic situation in which each convolutional layer expands the dimension of the data, which makes the contribution restrictive.

Some other research papers tackle the universal property of CNN with multi-dimensional output as a translation invariant function. Approaches that tackle the universal property of CNN with multi-dimensional output are investigating the approximation of the translation invariant function with convolutional neural networks [43, 30]. These papers consider the convolutional network as a function from $\mathbb{R}^d$ to $\mathbb{R}^d$. However, the invariance of the network inevitably prevents the use of practically used padding methods like zero padding. In addition, invariance fundamentally contradicts the universal property in the more general continuous function space.

In this regard, we studied the universality of the convolutional neural network consisting of the convolutional layer with zero padding. Unlike the previous methods that only consider scalar output or the translationally invariant functions, We directly tackle the universal property of CNN as a vector-to-vector function. Despite its dominant use in CNN, zero padding convolution has been outside the interests of the study because it deteriorates the invariance of the network. However, we revealed that zero padding is critical in achieving the universal property. More specifically, the universality occurs because zero padding interferes with invariance. We scrutinize the three-kernel convolutional neural network with zero padding and explore the minimal depth and width bound for the universal property. Our contributions are as follows:

- We proved that CNN has the universal property in the continuous function space as a function that preserves the shape of the input data.
- We found the optimal number of convolutional layers for a function with d-dimensional input to have the universal property.
- We proved that deep CNNs with $c_x + c_y + 2$ have the universal property, where c_x and c_y are the number of channels of the input and output data, respectively.

1.2 Recurrent Neural Network

Classical universal approximation theorems specialize in the representation power of shallow wide networks with bounded depth and unbounded width. Based on mounting empirical evidence that deep networks demonstrate better performance than wide networks, the construction of deep networks instead of shallow networks has gained considerable attention in recent literature. Consequently, researchers have started to analyze the universal approximation property of deep networks [5, 27, 34, 22]. Studies on MLP have shown that wide shallow networks require only two depths to have universality, while deep narrow networks require widths at least as their input dimension.

A wide network obtains universality by increasing its width even if the depth is only two [6, 18]. However, in the case of a deep network, there is a function for a narrow network that cannot be able to approximated, regardless of its depth [29, 33]. Therefore, clarifying the *minimum width* to guarantee universality is crucial, and studies are underway to investigate its lower and upper bounds, narrowing the gap.

Recurrent neural networks (RNNs) [35, 9] have been crucial for modeling complex temporal dependencies in sequential data. They have various applications in diverse fields, such as language modeling [31, 21], speech recognition [13, 3], recommendation systems [17, 41], and machine translation [2]. Deep RNNs are widely used and have been successfully applied in practical applications. However, their theoretical understanding remains elusive despite their intensive use. This deficiency in existing studies motivated our work.

In this thesis, we prove the universal approximation theorem of deep narrow RNNs and discover the upper bound of their minimum width. The target class consists of a sequence-to-sequence function that depends solely on past information. We refer to such functions as *past-dependent* functions. We provide the upper bound of the minimum width of the RNN for universality in the space of the past-

Network	Function class	Activation	Result
RNN	$C\left(\mathcal{K}, \mathbb{R}^{d_y}\right)^\dagger$	ReLU	$w_{\min} \leq d_x + d_y + 2$
		conti. nonpoly[1]	$w_{\min} \leq d_x + d_y + 3$
		conti. nonpoly[2]	$w_{\min} \leq d_x + d_y + 4$
	$L^p\left(\mathcal{K}, \mathbb{R}^{d_y}\right)^\dagger$	ReLU	$w_{\min} \leq \max\{d_x + 1, d_y\}$
		conti. nonpoly[2]	$w_{\min} \leq \max\{d_x + 1, d_y\} + 1$
LSTM	$C\left(\mathcal{K}, \mathbb{R}^{d_y}\right)^\dagger$		$w_{\min} \leq d_x + d_y + 3$
GRU	$C\left(\mathcal{K}, \mathbb{R}^{d_y}\right)^\dagger$		$w_{\min} \leq d_x + d_y + 3$
BRNN	$C\left(\mathcal{K}, \mathbb{R}^{d_y}\right)$	ReLU	$w_{\min} \leq d_x + d_y + 2$
		conti. nonpoly[1]	$w_{\min} \leq d_x + d_y + 3$
		conti. nonpoly[2]	$w_{\min} \leq d_x + d_y + 4$
	$L^p\left(\mathcal{K}, \mathbb{R}^{d_y}\right)$	ReLU	$w_{\min} \leq \max\{d_x + 1, d_y\}$
		conti. nonpoly[2]	$w_{\min} \leq \max\{d_x + 1, d_y\} + 1$

† requires the class to consists of past-dependent functions.
[1] requires an activation σ to be continuously differentiable at some point z_0 with $\sigma(z_0) = 0$ and $\sigma'(z_0) \neq 0$. tanh belongs here.
[2] requires an activation σ to be continuously differentiable at some point z_0 with $\sigma'(z_0) \neq 0$. A logistic sigmoid function belongs here.

Table 1.1: Summary of our results on the upper bound of the minimum width $w_{\min}$ of RNNs. In the table, $\mathcal{K}$ indicates a compact subset of $\mathbb{R}^{d_x}$ and $1 \leq p < \infty$. We abbreviate continuous to "conti" and denote the minimum width as $w_{\min}$.

dependent functions. Surprisingly, the upper bound is independent of the length of the sequence. This theoretical result highlights the suitability of the recurrent structure for sequential data compared with other network structures. Furthermore, our results are not restricted to RNNs; they can be generalized to variants of RNNs, including *long short-term memory* (LSTM), *gated recurrent units* (GRU), and *bidirectional RNNs* (BRNN). As corollaries of our main theorem, LSTM and GRU are shown to have the same universality and target class as an RNN. We also prove that the BRNN can approximate any sequence-to-sequence function in a continuous or L^p space under the respective norms. We also present the upper bound of the minimum width for these variants. Table 1.1 outlines our main results.

With a target class of functions that map a finite sequence $x \in \mathbb{R}^{d_x}$ to a finite sequence $y \in \mathbb{R}^{d_y}$, we prove the following:

- A deep RNN can approximate any past-dependent sequence-to-sequence continuous function with width $d_x + d_y + 2$ for the ReLU activation, $d_x + d_y + 3$ for tanh[1], and $d_x + d_y + 4$ for non-degenerating activations.

- A deep RNN can approximate any past-dependent L^p function ($1 \leq p < \infty$) with width $\max\{d_x + 1, d_y\}$ for the ReLU activation and $\max\{d_x + 1, d_y\} + 1$ for non-degenerating activations.

- A deep BRNN can approximate any sequence-to-sequence continuous function with width $d_x + d_y + 2$ for the ReLU activation, $d_x + d_y + 3$ for tanh[1], and $d_x + d_y + 4$ for non-degenerating activations.

- A deep BRNN can approximate any sequence-to-sequence L^p function ($1 \leq p < \infty$) with width $\max\{d_x + 1, d_y\}$ for the ReLU activation and $\max\{d_x + 1, d_y\} + 1$ for non-degenerating activations.

- A deep LSTM or GRU can approximate any past-dependent sequence-to-sequence continuous function with width $d_x + d_y + 3$ and L^p function with width $\max\{d_x + 1, d_y\} + 1$.

1.3 Related Works

We briefly review some of the results of studies on the universal approximation property. Studies have been conducted to determine whether a neural network

[1] Generally, non-degenerate σ with $\sigma(z_0) = 0$ requires the same minimal width as tanh.

can learn a sufficiently wide range of functions, that is, whether it has universal properties. In [6] and [18], the authors first proved that the most basic network, a simple two-layered MLP, can approximate arbitrary continuous functions defined on a compact set. Some follow-up studies have investigated the universal properties of other structures for a specific task, such as a convolutional neural network for image processing [48], an RNN for open dynamical systems [36, 15], and transformer networks for translation and speech recognition [44]. Particularly for RNNs, it is showed that open dynamical system with continuous state transition and output function can be approximated by a network with a wide RNN cell and subsequent linear layer in finite time [36]. Also, trajectory of the dynamical system can be reproduced with arbitrarily small errors up to infinite time, assuming a stability condition on long-term behavior [15].

While such prior studies mainly focused on wide and shallow networks, several studies have determined whether a deep narrow network with bounded width can approximate arbitrary functions [29, 14, 20, 22, 33]. Unlike the case of a wide network that requires only one hidden layer for universality, there exists a function that cannot be approximated by any network whose width is less than a certain threshold. More specifically, considering that d_x and d_y indicate the dimensions of the input and output vectors, respectively, the width d_x-1 is insufficient for an MLP to have universality in L^1 space if the activation function is ReLU [29]. In [14], there are negative and positive results which indicated that universality is not attained by width d_x, but width $d_x + d_y$ is sufficient to achieve universality in a continuous function space with ReLU activation. In [20, 22], the condition is generalized on the activation and proved that d_x is too narrow, but d_x+d_y+2 is sufficiently wide. The results show the lower bound d_x and upper bound d_x+d_y+C of the minimum width of the MLP, where C is a constant depending on the activation function. In [33], the exact value $\max\{d_x + 1, d_y\}$ required for universality is determined in L^p space with ReLU.

As described earlier, studies on deep narrow MLP have been actively conducted, but the approximation ability of deep narrow RNNs remains unclear. This is because the process by which the input affects the result is complicated compared with that of an MLP. The RNN cell transfers information to both the next time step in the same layer and the same time step in the next layer, which makes it difficult to investigate the minimal width. In this regard, we examined the structure of the RNN to apply the methodology and results from the study of MLPs to deep narrow RNNs.

On the other hand, research on the convolutional neural network as a general-purpose function is barely conducted. One of the research [48] studied the universal property of the convolutional neural network as a function from the vector to the scalar value. It tackles the network with a fully connected layer added to the last layer to make the network's output a scalar. Also, to employ the homomorphism between the composition of convolutional layers and the multiplication of polynomials, the paper assumed the impractical situation that data becomes longer as the data go through the network. On the other hand, we proved the case for a fully convolutional network that retains the shape of the input data to the output data. The authors of [43] focused on the periodic convolutional network's universal property as the translation equivariant function. However, the translation equivariance fundamentally contradicts the universal property as the general function from d-dimensional input data to the d-dimensional output data. Because the translation equivariance of the convolutional neural network is derived from cyclic padding, we need different padding, such as zero padding.

Chapter 2

The Universal Property of Convolutional Neural Network

2.1 Notion and Definition

We define notions and definitions that are used in the chapter. When we index the data in $\mathbb{R}^d$ or $\mathbb{R}^{\mathbb{Z}}$, we will use the subscript for indexing. For example, we express the components of $x \in \mathbb{R}^d$ as

$$x = (x_1, x_2, \ldots, x_d). \tag{2.1}$$

When we index the unique dimension called channel, we will use the superscript for indexing, that is, for $x \in \mathbb{R}^{c \times d}$,

$$x = (x^1, x^2, \ldots, x^c), \tag{2.2}$$

where $x^i \in \mathbb{R}^d$ for $i \in [1, c]$, and

$$x^i = (x^i_1, x^i_2, \dots, x^i_d). \tag{2.3}$$

The channel always comes first compared to other dimensions and is denoted as c or its variant. We also define the concatenation operation $\oplus$ along the channel as follows. For $x = (x^1, x^2, \dots, x^{c_1}) \in \mathbb{R}^{c_1 \times d}$ and $y = (y^1, y^2, \dots, y^{c_2}) \in \mathbb{R}^{c_2 \times d}$,

$$x \oplus y = (x^1, x^2, \dots, x^{c_1}, y^1, y^2, \dots, y^{c_2}) \in \mathbb{R}^{(c_1+c_2)\times d}. \tag{2.4}$$

We now define the mathematical contents used in the remaining sections.

- **Infinite-Length Convolution**: Let w be $w = (w_{-k}, w_{-k+1}, \dots, w_k) \in \mathbb{R}^{2k+1}$. Then an infinite-length convolution with kernel w is a map $f : \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^{\mathbb{Z}}$ defined as follows. For $x = (\dots, x_{-1}, x_0, x_1, \dots) \in \mathbb{R}^{\mathbb{Z}}$,

$$f_i(x) := \sum_{j=-k}^{k} w_j x_{i+j}, \tag{2.5}$$

  where $f(x) = (\dots, f_{-1}(x), f_0(x), f_1(x), \dots) \in \mathbb{R}^{\mathbb{Z}}$. We say that a convolution has a kernel size of $2k+1$.

- **Zero Padding Convolution**: Let $\iota : \mathbb{R}^d \to \mathbb{R}^{\mathbb{Z}}$ be a natural inclusion map. Formally, for $x = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d$,

$$\iota_i(x) := \begin{cases} x_i & \text{if } 1 \le i \le d, \\ 0 & \text{otherwise,} \end{cases} \tag{2.6}$$

  where $\iota(x) = (\dots, \iota_{-1}(x), \iota_0(x), \iota_1(x), \dots) \in \mathbb{R}^{\mathbb{Z}}$. And let $p_d : \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^d$ be a projection map; that is, for $x = (\dots, x_{-1}, x_0, x_1, \dots) \in \mathbb{R}^{\mathbb{Z}}$, $p_d(x)$ is defined as

$$p_d(x) := (x_1, x_2, \dots, x_d). \tag{2.7}$$

Let $w \in \mathbb{R}^{2k+1}$ be a kernel. Then zero padding convolution with kernel w is a function $f : \mathbb{R}^d \to \mathbb{R}^d$ is defined as

$$f := p_d \circ g \circ \iota, \tag{2.8}$$

where g is an infinite-length convolution with kernel w. We also define it as operation $\circledast$:

$$w \circledast x := f(x), \tag{2.9}$$

where g is an infinite-length convolution with kernel w. We can interpret the composition as constructing a temporary infinite-length sequence by filling zeros in the remaining components, conducting the convolution with kernel, and cutting off the unnecessary elements.

A zero padding convolution with kernel w is a linear transformation and hence can be expressed as matrix multiplication; $w \circledast x = Tx$ is satisfied for the following matrix $T \in \mathbb{R}^{d \times d}$:

$$T = \begin{bmatrix} w_0 & w_1 & \dots & w_k & & & & & & \\ w_{-1} & w_0 & \dots & w_{k-1} & w_k & & & & & \\ \vdots & & \ddots & \ddots & \ddots & & & & & \\ w_{-k} & w_{-k+1} & \dots & w_0 & w_1 & \dots & w_{k-1} & w_k & & \\ & w_{-k} & \dots & w_{-1} & w_2 & \dots & w_{k-2} & w_{k-1} & w_k & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & & & w_{-k} & w_{-k+1} & \dots & w_0 \end{bmatrix}. \tag{2.10}$$

We define the set of Toeplitz matrix as

$$To_d(s) := \left\{ (x_{i,j})_{1 \le i,j \le d} \in \mathbb{R}^{d \times d} \;\middle|\; x_{ij} = \begin{cases} w_{j-i} & \text{if } |i-j| \le s, \\ 0 & \text{otherwise.} \end{cases} \right\}. \tag{2.11}$$

Also, define $U_t = (u_{i,j})_{1 \le i,j \le d}$ as

$$u_{i,j} = \begin{cases} 1 & \text{if } i - j = t, \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

By definition, U_0 is an identity matrix, and U_t and U_{-t} have a transpose relationship with each other; $U_t^T = U_{-t}$. The set $\{U_{-s}, U_{-s+1}, \dots U_s\}$ is the basis of the set of Toeplitz matrices $To_d(s)$. Zero padding convolution with kernel $w = (w_{-s}, w_{-s+1}, \dots, w_s)$ can be represented as

$$w \circledast x = \sum_{i=-s}^{s} w_i U_{-i}. \tag{2.13}$$

Obviously, $(U_1)^t = U_t$, and $(U_{-1})^t = U_{-t}$ for $t \geq 0$. Also, it is convenient to interpret the matrix multiplication in the following way. Let A be a matrix or a column vector. Then, U_tA and $U_{-t}A$ move A downward t rows and upward t rows, respectively. Similarly, AU_t and AU_{-t} move A to the left by t columns and right by t columns, respectively. We also define $E_{n,m} := (e_{i,j})_{1\leq i,j\leq d}$ as

$$e_{i,j} = \begin{cases} 1 & \text{if } i = n \text{ , and } j = m, \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

To deal with the composition of convolutions, we define S_N as follows.

$$S_N := \left\{ \sum_{i=1}^{n} \prod_{j=1}^{N} T_{i,j} \,\middle|\, T_{i,j} \in To_d(1), n \in \mathbb{N} \right\}. \tag{2.15}$$

S_N is a vector space of matrix representations of linear transformations that a linear three-kernel N-layered CNN can express.

- **Zero Padding Convolutional Layer:** A convolutional layer with c_1 input channels and c_2 output channels is a map $f : \mathbb{R}^{c_1\times d} \to \mathbb{R}^{c_2\times d}$. For each $1 \leq i \leq c_2$ and $1 \leq j \leq c_1$, there exist zero padding convolutions with kernel $w_{i,j} \in \mathbb{R}^{2k+1}$ and bias $\delta_i \in \mathbb{R}$ so that for $x = (x^1, x^2, \ldots, x^{c_1}) \in \mathbb{R}^{c_1\times d}$,

$$f^i(x) := \sum_{j=1}^{c_1} w_{i,j} \circledast x^j + \delta_i \mathbf{1}_d, \tag{2.16}$$

where $f(x) = (f^1(x), f^2(x), \ldots, f^{c_2}(x))$. We extend the operation $\circledast$ to the multiplication between the vector-valued matrix. Let $M_{n,m}(\mathbb{R}^d)$ be the $n \times m$ matrix whose components are d-dimensional vectors in $\mathbb{R}^d$. Then for $A =$

$(a_{i,j})_{1\leq i\leq n,1\leq j\leq m} \in M_{n,m}(\mathbb{R}^{2k+1})$ and $B = (b_{j,k})_{1\leq j\leq m,1\leq k\leq l} \in M_{m,l}(\mathbb{R}^d)$, we denote matrix multiplication $\circledast$ between A and B as

$$C := A \circledast B, \tag{2.17}$$

where $C = (c_{i,k})_{1\leq i\leq n,1\leq k\leq l} \in M_{n,l}(\mathbb{R}^d)$, and $c_{i,k}$ is calculated as

$$c_{i,k} := \sum_{j=1}^{m} a_{i,j} \circledast b_{j,k}. \tag{2.18}$$

Zero padding convolutional layer can be interpreted as a matrix multiplication between weight matrix $W = (w_{i,j})_{1\leq i\leq c_2,1\leq j\leq c_1} \in M_{c_2,c_1}(\mathbb{R}^d)$ and input vector $X = (x^j)_{1\leq j\leq c_1} \in M_{c_1,1}(\mathbb{R}^d)$ and bias summation.

$$\begin{bmatrix} f^1 \\ f^2 \\ \vdots \\ f^{c_2} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,c_1} \\ w_{2,1} & w_{2,2} & \dots & w_{2,c_1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{c_2,1} & w_{c_2,2} & \dots & w_{c_2,c_1} \end{bmatrix} \circledast \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^{c_1} \end{bmatrix} + \begin{bmatrix} \delta_1 \mathbf{1}_d \\ \delta_2 \mathbf{1}_d \\ \vdots \\ \delta_{c_2} \mathbf{1}_d \end{bmatrix}. \tag{2.19}$$

- **Activation Function:** An activation function σ is a scalar function $\sigma : \mathbb{R} \to \mathbb{R}$. We extend the function component-wise to the multivariate versions $\sigma_d : \mathbb{R}^d \times \mathbb{R}^d$ and $\sigma_{c,d} : \mathbb{R}^{c\times d} \times \mathbb{R}^{c\times d}$. Specifically, for $x \in \mathbb{R}^d$,

$$\sigma_d(x) := (\sigma(x_1), \sigma(x_2), \dots, \sigma(x_d)), \tag{2.20}$$

where $x = (x_1, x_2, \dots, x_d)$. And for $x = (x^1, x^2, \dots, x^c) \in \mathbb{R}^{c\times d}$ and $x^i =$

$(x_1^i, x_2^i, \ldots, x_d^i) \in \mathbb{R}^d$,

$$(\sigma_{c,d}(x))_j^i = \sigma(x_j^i) \ \text{ for } 1 \leq i \leq c, 1 \leq j \leq d. \tag{2.21}$$

We will slightly abuse notation so that σ means σ, σ_d, and $\sigma_{c,d}$, depending on the context.

We also define a modified version of activation function that selectively applies an activation function to each channel by modifying the activation function as follows. For $I \subset [1, c]$, define $\widetilde{\sigma}_I : \mathbb{R}^{c \times d} \to \mathbb{R}^{c \times d}$ as follows. If $x = (x^1, x^2, \ldots, x^c)$ and $x^i \in \mathbb{R}^d$,

$$\widetilde{\sigma}_I^i(x) = \begin{cases} \sigma(x^i) & \text{if } i \in I, \\ x^i & \text{otherwise,} \end{cases} \tag{2.22}$$

where $\widetilde{\sigma}_I = (\widetilde{\sigma}_I^1, \widetilde{\sigma}_I^2, \ldots, \widetilde{\sigma}_I^c)$.

- **Convolutional Neural Network:** An N-layered convolutional neural network with c input channels and c' output channels is a map $f : \mathbb{R}^{c_0 \times d} \to \mathbb{R}^{c_N \times d}$ that is constructed by following N convolutional layers and the activation function. For the channel sizes $c_0 = c, c_1, \ldots, c_N = c'$, there exist convolutional layers $C_i : \mathbb{R}^{c_{i-1} \times d} \to \mathbb{R}^{c_i \times d}$, and f is defined as follows.

$$f := C_N \circ \sigma \circ C_{N-1} \circ \cdots \circ \sigma \circ C_1. \tag{2.23}$$

We denote the channel sizes of the convolutional layer as $c_0 - c_1 - \cdots - c_n$. Then, we define $\Sigma_{c,c'}^N$ as the set of the convolutional neural networks with c input channels and c' output channels:

$$\begin{aligned}\Sigma^N_{c,c'} := \{ C_N \circ \sigma \circ C_{N-1} \circ \cdots \circ \sigma \circ C_1 : \mathbb{R}^{c \times d} \to \mathbb{R}^{c' \times d} \ \Big| \\ c_1, c_2, \dots c_{N-1} \in \mathbb{N} \text{ ,where } c = c_0, c' = c_N \text{ and} \\ C_i : \mathbb{R}^{c_{i-1} \times d} \to \mathbb{R}^{c_i \times d} \text{ are the 3-kernel convolutional layers} \}.\end{aligned} \tag{2.24}$$

If we need to indicate the activation function explicitly, we denoted $\Sigma^N_{c,c'}$ as ${}^{\sigma}\Sigma^N_{c,c'}$. Also, define $\sigma(\Sigma^N_{c,c'})$ as

$$\sigma\left(\Sigma^N_{c,c'}\right) := \left\{ \sum_{i=1}^{n} a_i(\sigma \circ f_i) \ \middle|\ f_i \in \Sigma^N_{c,c'}, a_i \in \mathbb{R}, n \in \mathbb{N}_0 \right\}. \tag{2.25}$$

2.2 Main Theorem

2.2.1 Problem Formulation

The universal property of CNN, which we will discuss in this chapter, is whether a continuous function from $\mathbb{R}^{c \times d}$ to $\mathbb{R}^{c' \times d}$ can be uniformly approximated by convolutional neural networks. Let $C(X, Y)$ be a space of continuous function from X to Y. Then we define the norm in $C(K, \mathbb{R}^{c' \times d})$ for each compact subset $K \subset \mathbb{R}^{c \times d}$ as follows:

$$||f - g||_{C^\infty(K)} = \sup_{x \in K} ||f(x) - g(x)||_2. \tag{2.26}$$

What we want to show in Section 2.2.3 is under what condition, the closure with respect to $C^\infty(K)$ norm satisfy the following statement,

$$\overline{\Sigma^N_{c,c'}} = C(K, \mathbb{R}^{c' \times d}). \tag{2.27}$$

And in Section 2.2.4, we will show that convolutional neural networks with bounded width are also dense in $C(K, \mathbb{R}^{c' \times d})$ with respect to $C^\infty(K)$ norm.

2.2.2 Lemmas

Now we present proofs for theorems. Before we get into the main theorems, we first prove the lemma that will be used for proofs.

Lemma 2.1. *The following statements are satisfied.*

1. *$\Sigma^N_{c,c'}$ is closed under concatenation. In other words, for $f_1 \in \Sigma^N_{c,c'}$ and $f_2 \in \Sigma^N_{c,c''}$, the function f is defined as $f(x) := f_1(x) \oplus f_2(x) \in \mathbb{R}^{(c'+c'')\times d}$. Then, $f \in \Sigma^N_{c,c'+c''}$.*

2. *$\Sigma^N_{c,c'}$ and $\sigma(\Sigma^N_{c,c'})$ are vector spaces.*

3. *For a C^∞ activation function σ, $\overline{{}^\sigma\Sigma^N_{c,c'}}$ is closed under partial differentiation; for C^∞ function $f(x,\theta)$ and $f_\theta(x) := f(x,\theta)$, if $f_\theta(x) \in \overline{\Sigma^N_{c,c'}}$, then, $\frac{\partial}{\partial\theta}(f_\theta) \in \overline{\Sigma^N_{c,c'}}$. Also, $\overline{\sigma(\Sigma^N_{c,c'})}$ is closed under partial differentiation.*

4. *For $f \in \overline{\sigma(\Sigma^N_{c,c'})}$ and a convolutional layer C with c' input channels and c'' output channels, $C \circ f \in \overline{\Sigma^{N+1}_{c,c''}}$.*

Proof. 1. Let f_1 with channel sizes $c - c_1 - c_2 - \cdots - c_{N-1} - c'$ be

$$f_1 := C_N \circ \sigma \circ C_{N-1} \circ \cdots \circ \sigma \circ C_1, \tag{2.28}$$

and f_2 with channel sizes $c - c'_1 - c'_2 - \cdots - c'_{N-1} - c''$ be

$$f_2 := C'_N \circ \sigma \circ C'_{N-1} \circ \cdots \circ \sigma \circ C'_1. \tag{2.29}$$

As in Equation (2.19), we can express C_i as

$$C_i(x) = W_i \circledast x + \boldsymbol{\delta}_i, \tag{2.30}$$

where W_i is the $c_i \times c_{i-1}$ matrix of kernels, and $\boldsymbol{\delta}_i$ is the vector of length c_i consisting of d-dimensional vectors. Similarly, we can denote C'_i as

$$C'_i(x) = W'_i \circledast x + \boldsymbol{\delta'}_i. \tag{2.31}$$

Then we can define the concatenation for $i = 2, 3, \ldots, N$ as

$$C''_i(x \oplus y) := \begin{bmatrix} W_i & \\ & W'_i \end{bmatrix} \circledast \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \boldsymbol{\delta}_i \\ \boldsymbol{\delta'}_i \end{bmatrix} = C_1(x) \oplus C_2(y). \tag{2.32}$$

Also, define C''_1 as

$$C''_1(x) := \begin{bmatrix} W_1 \\ W'_1 \end{bmatrix} \circledast x + \begin{bmatrix} \boldsymbol{\delta}_1 \\ \boldsymbol{\delta'}_1 \end{bmatrix} = C_1(x) \oplus C'_1(x). \tag{2.33}$$

Then, we can construct f with channel sizes $c - (c_1 + c'_1) - (c_2 + c'_2) - \cdots - (c_{N-1} + c'_{N-1}) - (c' + c'')$ as

$$f := C''_N \circ \sigma \circ C''_{N-1} \circ \cdots \circ \sigma \circ C''_1. \tag{2.34}$$

$$f(x) = C''_N \circ \sigma \circ C''_{N-1} \circ \cdots \circ \sigma \circ C''_1(x) \tag{2.35}$$

$$= C''_N \circ \sigma \circ C''_{N-1} \circ \cdots \circ (\sigma \circ C_1(x) \oplus \sigma \circ C_2(x)) \tag{2.36}$$

$$= (C_N \circ \sigma \circ \cdots \circ \sigma \circ C_1(x)) \oplus (C'_N \circ \sigma \circ \cdots \circ \sigma \circ C'_1(x)) \tag{2.37}$$

$$= f_1(x) \oplus f_2(x), \tag{2.38}$$

which completes the proof.

2. For the arbitrary $f_1, f_2 \in \Sigma^N_{c,c'}$, express f_1 and f_2 as $f_1 := C_N \circ \sigma \circ C_{N-1} \circ \cdots \circ \sigma \circ C_1$ and $f_2 := C'_N \circ \sigma \circ C'_{N-1} \circ \cdots \circ \sigma \circ C'_1$. Except for the axiom that $\Sigma^N_{c,c'}$ is closed under addition, other axioms can be shown simply by giving proper operations to the last layer. For example, replacing C_N with $-C_N$ gives the inverse element of f_1. For the axiom that $\Sigma^N_{c,c'}$ is closed under addition, construct g as the concatenation of $g_1 := C_{N-1} \circ \cdots \circ \sigma \circ C_1$ and $g_2 := C'_{N-1} \circ \cdots \circ \sigma \circ C'_1$. For the C_N and C'_N expressed as

$$C_N(x) = W \circledast x + \boldsymbol{\delta}, \tag{2.39}$$

and

$$C'_N(x) = W' \circledast x + \boldsymbol{\delta'}, \tag{2.40}$$

we can construct the convolutional layer C''_N, which satisfies

$$C''_N(x \oplus y) = \begin{bmatrix} W & W' \end{bmatrix} \circledast \begin{bmatrix} x \\ y \end{bmatrix} + (\boldsymbol{\delta} + \boldsymbol{\delta'}) = C_N(x) + C'_N(y). \tag{2.41}$$

Then,

$$C''_N \circ \sigma \circ g(x) = C''_N \circ \sigma g_1(x) \oplus g_2(x)$$
$$= C_N \circ \sigma \circ g_1(x) + C'_N \circ \sigma \circ g_2(x) = f_1(x) + f_2(x). \quad (2.42)$$

Thus, $f_1 + f_2 \in \Sigma^N_{c,c'}$, and $\Sigma^N_{c,c'}$ is a vector space.

For $\sigma(\Sigma^N_{c,c'})$, it is obvious from the definition of $\sigma(\Sigma^N_{c,c'})$.

3. Because $\Sigma^N_{c,c'}$ is a vector space, $\frac{f_{\theta+\epsilon}(x)-f_\theta(x)}{\epsilon} \in \Sigma^N_{c,c'}$. And because $||\frac{f_{\theta+\epsilon}(x)-f_\theta(x)}{\epsilon} - \frac{\partial}{\partial\theta} f_\theta(x)|| < o(\epsilon)\sup_\theta ||\frac{\partial^2}{\partial^2\theta} f_\theta(x)||$, it uniformly converges to zero; thus, $\frac{\partial}{\partial\theta} f_\theta(x) \in \overline{{}^\sigma\Sigma^N_{c,c'}}$. Similar argument holds for $\overline{\sigma(\Sigma^N_{c,c'})}$.

4. For $g \in \overline{\sigma(\Sigma^N_{c,c'})}$, there exist $g_i \in \sigma(\Sigma^N_{c,c'})$, such that $g_i \xrightarrow{i\to\infty} g$. Then, we have

$$g_i = \sum_{j=1}^{n_i} a_{i,j}(\sigma \circ g_{i,j}), \quad (2.43)$$

for $g_{i,j} \in \Sigma^N_{c,c'}$ and $a_{i,j} \in \mathbb{R}$. Decompose C into $C = L + \boldsymbol{\delta}$ where L is the linear transformation and $\boldsymbol{\delta}$ is the bias:

$$C \circ g_i = (L + \boldsymbol{\delta}) \circ \sum_{j=1}^{n_i} a_{i,j}(\sigma \circ g_{i,j}) = \boldsymbol{\delta} + \sum_{j=1}^{n_i} a_{i,j} L \circ \sigma \circ g_{i,j} \in \Sigma^{N+1}_{c,c''}, \quad (2.44)$$

because $\Sigma^{N+1}_{c,c''}$ is a vector space. If $\{g_i\}_{i\in\mathbb{N}}$ uniformly converges to g, $\{C \circ g_i\}_{i\in\mathbb{N}}$ uniformly converges to $C \circ g$ for the continuous function C in the compact space, and it completes the proof.

$\square$

Lemma 2.2. *Consider the activation function σ, which is the C^1-function near α*

and $\sigma'(\alpha) \neq 0$. Then, for zero padding convolutional layers C_1, C_2 and a positive number $\epsilon \in \mathbb{R}_+$, there exist zero padding convolutional layers C_1', C_2' with same input and output channels to C_1, C_2, respectively, such that

$$||C_2 \circ \widetilde{\sigma}_I \circ C_1 - C_2' \circ \sigma \circ C_1'||_{C^\infty(K)} < \epsilon, \tag{2.45}$$

where $I \subset [1, c_2]$, and c_2 is the number of output channels of C_1.

Proof. Let C_1 and C_2 be $C_1 : \mathbb{R}^{c_1 \times d} \to \mathbb{R}^{c_2 \times d}$ and $C_2 : \mathbb{R}^{c_2 \times d} \to \mathbb{R}^{c_3 \times d}$. C_1 has kernels $w^1_{j,i}$ and biases δ^1_j for $i \in [1, c_1]$ and $j \in [1, c_2]$, and C_2 has kernels $w^2_{k,j}$ and biases δ^2_k for $j \in [1, c_2]$ and $k \in [1, c_3]$. Then, define C_1' with kernels $w'^1_{j,i}$ and biases δ'^1_j and C_2' with kernels $w'^2_{k,j}$ and biases δ'^2_k as follows:

$$w'^1_{j,i} = \begin{cases} w^1_{j,i} & \text{if } j \in I, \\ \frac{w^1_{j,i}}{N} & \text{otherwise,} \end{cases} \qquad \delta'^1_j = \begin{cases} \delta^1_j & \text{if } j \in I, \\ \alpha + \frac{\delta^1_j}{N} & \text{otherwise }, \end{cases} \tag{2.46}$$

and

$$w'^2_{k,j} = \begin{cases} w^2_{k,j} & \text{if } j \in I, \\ \frac{N}{\sigma'(\alpha)} w^2_{k,j} & \text{otherwise,} \end{cases} \qquad \delta'^2_k = \frac{N\sigma(\alpha)}{\sigma'(\alpha)} + \delta^2_k. \tag{2.47}$$

Then, f_k, the k-th component of $C_2' \circ \sigma \circ C_1'$, becomes

$$f_k(x) := \sum_{j=1}^{c_2} w'^2_{k,j} \sigma \left(\sum_{i=1}^{c_1} w'^1_{j,i} \circledast (x^j) + \delta'^1_j \mathbf{1}_d \right) + \delta'^2_k \mathbf{1}_d \tag{2.48}$$

$$= \sum_{j \in I} w'^2_{k,j} \sigma \left(\sum_{i=1}^{c_1} w'^1_{j,i} \circledast (x^j) + \delta'^1_j \mathbf{1}_d \right) \tag{2.49}$$

$$+ \sum_{j \notin I} w'^2_{k,j} \sigma \left(\sum_{i=1}^{c_1} w'^1_{j,i} \circledast (x^j) + \delta'^1_j \mathbf{1}_d \right) + \delta'^2_k \mathbf{1}_d \tag{2.50}$$

$$= \sum_{j \in I} w^2_{k,j} \sigma \left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d \right) \tag{2.51}$$

$$+ \sum_{j \notin I} \frac{N}{\sigma'(\alpha)} w^2_{k,j} \sigma \left(\sum_{i=1}^{c_1} \frac{w^1_{j,i}}{N} \circledast (x^j) + \frac{\delta^1_j}{N} + \alpha \right) + \frac{N\sigma(\alpha)}{\sigma'(\alpha)} + \delta^2_k \mathbf{1}_d. \tag{2.52}$$

And the k-th component of $C_2 \circ \widetilde{\sigma}_I \circ C_1$, g_k, is

$$g_k(x) = \sum_{j=1}^{c_2} w^2_{k,j} \widetilde{\sigma}_I \left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d \right) + \delta^2_k \mathbf{1}_d \tag{2.53}$$

$$= \sum_{j \in I} w^2_{k,j} \widetilde{\sigma}_I \left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d \right) \tag{2.54}$$

$$+ \sum_{j \notin I} w^2_{k,j} \widetilde{\sigma}_I \left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d \right) + \delta^2_k \mathbf{1}_d \tag{2.55}$$

$$= \sum_{j \in I} w^2_{k,j} \sigma \left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d \right) \tag{2.56}$$

$$+ \sum_{j \notin I} w^2_{k,j} \left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d \right) + \delta^2_k \mathbf{1}_d. \tag{2.57}$$

Then $f_k - g_k$ becomes

$$g_k(x) - f_k(x) \tag{2.58}$$
$$= \sum_{j\in I} w^2_{k,j}\sigma\left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d\right) \tag{2.59}$$
$$+ \sum_{j\notin I} w^2_{k,j}\left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d\right) + \delta^2_k \mathbf{1}_d \tag{2.60}$$
$$- \sum_{j\in I}\left(w^2_{k,j}\sigma\left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d\right)\right) \tag{2.61}$$
$$- \sum_{j\notin I}\left(\frac{N}{\sigma'(\alpha)} w'^2_{k,j}\sigma\left(\sum_{i=1}^{c_1} \frac{w^1_{j,i}}{N} \circledast (x^j) + \frac{\delta^1_j \mathbf{1}_d}{N} + \alpha\right)\right) - \frac{N\sigma(\alpha)}{\sigma'(\alpha)} - \delta^2_k \mathbf{1}_d \tag{2.62}$$
$$= \sum_{j\notin I} w^2_{k,j}\left(\sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d\right) + \delta^2_k \mathbf{1}_d \tag{2.63}$$
$$- \sum_{j\notin I}\left(\frac{N}{\sigma'(\alpha)} w^2_{k,j}\sigma\left(\sum_{i=1}^{c_1} \frac{w^1_{j,i}}{N} \circledast (x^j) + \frac{\delta^1_j \mathbf{1}_d}{N} + \alpha\right)\right) - \frac{N\sigma(\alpha)}{\sigma'(\alpha)} - \delta^2_k \mathbf{1}_d. \tag{2.64}$$

Let u_j be $u_j := \sum_{i=1}^{c_1} w^1_{j,i} \circledast (x^j) + \delta^1_j \mathbf{1}_d$. Then,

$$g_k(x) - f_k(x) = \sum_{j\notin I} w^2_{k,j}\left(u_j - \frac{N}{\sigma'(\alpha)}\sigma\left(\frac{u_j}{N} + \alpha\right) - \frac{N\sigma(\alpha)}{\sigma'(\alpha)}\right) \tag{2.65}$$
$$= \sum_{j\notin I} w^2_{k,j}\frac{u_j^2}{\sigma'(\alpha)} o(\frac{1}{N}) \xrightarrow{\text{N}\to\infty} 0. \tag{2.66}$$

The convergence is uniform because x is in the compact domain K; thus, u_j is uniformly bounded for all x. $\square$

Lemma 2.3. *For the Lipschitz continuous activation function σ, $N \geq 2$, the channel sizes $c_0 - c_1 - \cdots - c_N$, indexes $I_i \subset [1, c_i]$, and the convolutional layers C_i with c_{i-1} input channels and c_i output channels, define the convolutional neural network*

f as

$$f := C_N \circ \widetilde{\sigma}_{I_{N-1}} \circ C_{N-1} \circ \cdots \circ \widetilde{\sigma}_{I_1} \circ C_1. \tag{2.67}$$

Then, there exists $g \in {}^{\sigma}\Sigma^N_{c,c'}$ defined as

$$g := C'_N \circ \sigma \circ C'_{N-1} \circ \cdots \circ \sigma \circ C'_1, \tag{2.68}$$

such that

$$\|f - g\|_{C^\infty(K)} < \epsilon, \tag{2.69}$$

where C'_i has c_{i-1} input channels and c_i output channels.

Proof. Use the mathematical induction on N. By Lemma 2.2, the induction hypothesis is satisfied for the case $N = 2$. Assume that the induction hypothesis is satisfied for the case $N = N_0$. For the case $N = N_0 + 1$, consider the function f_{N_0+1} defined as

$$f_{N_0+1} = C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}} \circ C_{N_0} \circ \cdots \circ \widetilde{\sigma}_{I_1} \circ C_1. \tag{2.70}$$

Then, for $f_{N_0} := C_{N_0} \circ \widetilde{\sigma}_{I_{N_0}-1} \circ \cdots \circ C_1$,

$$f_{N_0+1} = C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}} \circ f_{N_0}. \tag{2.71}$$

By the induction hypothesis, there exists $g \in {}^{\sigma}\Sigma^{N_0}_{c,c'}$, such that

$$\|f_{N_0} - g\|_{C^\infty(K)} < \frac{\epsilon}{2l}. \tag{2.72}$$

where l denote the Lipschitz constant of $C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}}$. Then,

$$\|f_{N_0+1} - C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}} \circ g\|_{C^\infty(K)} < \frac{\epsilon}{2}. \tag{2.73}$$

Denote g as

$$g = C'_{N_0} \circ \sigma \circ \cdots \circ \sigma \circ C'_1. \tag{2.74}$$

By Lemma 2.2, there exist convolutional layers C''_{N_0+1} and C''_{N_0}, such that

$$\|C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}} \circ C'_{N_0} - C''_{N_0+1} \circ \sigma \circ C''_{N_0}\|_{C^\infty(K')} < \frac{\epsilon}{2}. \tag{2.75}$$

where K' is the compact space $K' = \sigma \circ C'_{N_0-1} \circ \cdots \circ \sigma \circ C'_1(K)$. Define $h \in {}^{\sigma}\Sigma^{N_0}_{c,c'}$ as

$$h := C''_{N_0+1} \circ \sigma \circ C''_{N_0} \circ \sigma \circ C'_{N_0-1} \circ \cdots \circ C'_1. \tag{2.76}$$

Then, the following equation is satisfied:

$$\|C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}} \circ g - h\|_{C^\infty(K)} < \frac{\epsilon}{2}. \tag{2.77}$$

To sum up,

$$\begin{aligned} &\|f_{N_0+1} - h\|_{C^\infty(K)} < \\ &\quad \|f_{N_0+1} - C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}} \circ g\|_{C^\infty(K)} + \|C_{N_0+1} \circ \widetilde{\sigma}_{I_{N_0}} \circ g - h\|_{C^\infty(K)} < \epsilon. \end{aligned} \tag{2.78}$$

Therefore, the induction hypothesis is satisfied for $N = N_0 + 1$, and it completes the proof. □

Corollary 2.4. *For the Lipschitz continuous activation function σ and $N \geq 2$, ${}^{Id}\Sigma^N_{c,c'}$ is the subset of $\overline{{}^{\sigma}\Sigma^N_{c,c'}}$ as functions defined on the compact set K where Id is the identity function; that is, ${}^{Id}\Sigma^N_{c,c'} \subset \overline{{}^{\sigma}\Sigma^N_{c,c'}}$.*

Lemma 2.3 and Corollary 2.4 imply that we can freely exchange the activation function to the identity.

2.2.3 The Minimum Depth for the Universal Property of Convolutional Neural Network

In this section, we showed the minimum depth for the three-kernel convolutional neural network to be universal. Unlike MLP, which only needs a two-layered network to get universality, CNN requires a much deeper minimum depth. This is because the receptive field, the range of the input component which affects the specific output component, is restricted by the convolution using the kernel. In the case of a convolutional layer with a kernel size of three, each output receives input from left and right one component. Therefore, when considering the convolutional neural network constructed by composing these N layers of convolutional layers, the input can take values from the left and right N components. Therefore, obviously, in the case of a function with d-dimensional input and output, at least $d-1$ layers must be used for the first component of the output to receive the last component of the input. Therefore, for a CNN with kernel size three to have the universal property, at least $d-1$ layers are required. The following proposition shows that the minimum depth $d-1$ is insufficient for the case of $d=3$.

Proposition 2.5. *If a compact domain $K \in \mathbb{R}^c$ contains an open subset near the origin, three-kernel two-layered CNN does not have the universal property in K when $d=3$; that is, $\overline{\Sigma^2_{c,c'}} \neq C(K, \mathbb{R}^{c'})$.*

Proof. For a 3-dimensional input, consider the case where the numbers of input and output channels are one, and the number of intermediate channels is n. Then, for a convolutional layer C_1 with kernels (a^i_{-1}, a^i_0, a^i_1) and biases δ_i and a convolutional layer C_2 with kernels (b^i_{-1}, b^i_0, b^i_1) and biases δ_0, the entire CNN $f = (f_1, f_2, f_3) := C_2 \circ \sigma \circ C_1$ satisfies the following equations.

$$f_1(x_1, x_2, x_3) = \sum_{i=1}^{n} b^i_0 \sigma(a^i_0 x_1 + a^i_1 x_2 + \delta_i) + b^i_1 \sigma(a^i_{-1} x_1 + a^i_0 x_2 + a^i_1 x_3 + \delta_i) + \delta_0, \quad (2.79)$$

$$\begin{aligned} f_2(x_1, x_2, x_3) = \sum_{i=1}^{n} & b^i_{-1} \sigma(a^i_0 x_1 + a^i_1 x_2 + \delta_i) \\ & + b^i_0 \sigma(a^i_{-1} x_1 + a^i_0 x_2 + a^i_1 x_3 + \delta_i) + b^i_1 \sigma(a^i_{-1} x_2 + a^i_0 x_3 + \delta_i) + \delta_0, \end{aligned} \quad (2.80)$$

$$\begin{aligned} f_3(x_1, x_2, x_3) = \sum_{i=1}^{n} & b^i_{-1} \sigma(a^i_{-1} x_1 + a^i_0 x_2 + a^i_1 x_3 + \delta_i) \\ & + b^i_0 \sigma(a^i_{-1} x_2 + a^i_0 x_3 + \delta_i) + \delta_0. \end{aligned} \quad (2.81)$$

Then, the following equation holds.

$$f_1(x, y, 0) - f_2(0, x, y) \quad (2.82)$$

$$= \left(\sum_{i=1}^{n} b^i_0 \sigma(a^i_0 x + a^i_1 y + \delta_i) + b^i_1 \sigma(a^i_{-1} x + a^i_0 y + \delta_i) \right) \quad (2.83)$$

$$- \left(\sum_{i=1}^{n} b^i_{-1} \sigma(a^i_1 x + \delta_i) + b^i_0 \sigma(a^i_0 x + a^i_1 y + \delta_i) + b^i_1 \sigma(a^i_{-1} x + a^i_0 y + \delta_i) \right) \quad (2.84)$$

$$= - \sum_{i=1}^{n} b^i_{-1} \sigma(a^i_1 x + \delta_i). \quad (2.85)$$

Thus, $f_1(x, y, 0) - f_2(0, x, y)$ becomes the function of x. Let it

$$h(x) := f_1(x, y, 0) - f_2(0, x, y). \quad (2.86)$$

Also, define $g = (g_1, g_2, g_3) : \mathbb{R}^3 \to \mathbb{R}^3$ as

$$g(x_1, x_2, x_3) := (x_2, 0, 0). \tag{2.87}$$

Let K contains the open rectangle $(-\epsilon_0, \epsilon_0)^3$. Then, the following equation is satisfied for arbitrary $x, y \in (-\epsilon_0, \epsilon_0)$.

$$|(f_1 - g_1)(x, y, 0) - (f_2 - g_2)(0, x, y)| = |y - h(x)|. \tag{2.88}$$

If $g \in \overline{\Sigma^2_{c,c'}}$, there exists f such that,

$$||f - g||_{C^\infty(K)} < \frac{\epsilon_0}{4}, \tag{2.89}$$

which implies that $|(f_1 - g_1)(x, y, 0)| < \frac{\epsilon_0}{4}$ and $|(f_2 - g_2)(0, x, y)| < \frac{\epsilon_0}{4}$ for arbitrary $x, y \in (-\epsilon_0, \epsilon_0)$. However,

$$\begin{aligned} |y - h(x)| =& |(f_1 - g_1)(x, y, 0) - (f_1 - g_2)(0, x, y)| \\ &< |(f_1 - g_1)(x, y, 0)| + |(f_1 - g_2)(0, x, y)| < \frac{\epsilon_0}{2}, \end{aligned} \tag{2.90}$$

for arbitrary $x, y \in [-\epsilon_0, \epsilon_0]$, which becomes a contradiction, and it completes the proof. □

Now we provide the main proposition of the chapter. Before we go further, we will prove some important lemmas.

Lemma 2.6. *For $i \in [1, n]$, $l \in \mathbb{N}$, and a non-polynomial C^∞ activation function*

σ, if $A_i \in \overline{\Sigma^l_{c,1}}$, then the following relation holds:

$$\prod_{i=1}^{n} A_i \in \overline{\sigma(\Sigma^l_{c,1})}, \tag{2.91}$$

where the product on the left hand side means the Hadamard product of the vector-valued functions.

Proof. Let $a_i \in \mathbb{R}$ for $i \in [1, n]$. Because $\overline{\Sigma^l_{c,1}}$ is a vector space by Lemma 2.1, and $\delta \mathbf{1}_d \in \overline{\Sigma^l_{c,1}}$, the linear summation also in $\overline{\Sigma^l_{c,1}}$:

$$f := \sum_{i=1}^{n} a_i A_i + \delta \mathbf{1}_d \in \overline{\Sigma^l_{c,1}}. \tag{2.92}$$

By definition of $\overline{\sigma(\Sigma^l_{c,1})}$,

$$\sigma\left(\sum_{i=1}^{n} a_i A_i + \delta \mathbf{1}_d\right) \in \overline{\sigma(\Sigma^l_{c,1})}. \tag{2.93}$$

By Lemma 2.1, $\overline{\sigma(\Sigma^l_{c,1})}$ is closed under the partial differentiation with respect to the parameters. Therefore, we have

$$\left(\prod_{i=1}^{n} \frac{\partial}{\partial a_i}\right)\left[\sigma\left(\sum_{i=1}^{n} a_i A_i + \delta \mathbf{1}_d\right)\right] \in \overline{\sigma(\Sigma^l_{c,1})}. \tag{2.94}$$

And the partial differentiation is calculated as the Hadamard product:

$$\left(\prod_{i=1}^{n} \frac{\partial}{\partial a_i}\right)\left[\sigma\left(\sum_{i=1}^{n} a_i A_i + \delta \mathbf{1}_d\right)\right] = \prod_{i=1}^{n} A_i \sigma^{(n)}(f) \in \overline{\sigma(\Sigma^l_{c,1})}. \tag{2.95}$$

Because σ is the non-polynomial function, there exist δ_0 such that $\sigma^{(n)}(\delta_0) \neq 0$. By substituting all a_i to zero and δ to δ_0, we get

$$\prod_{i=1}^{n} A_i \sigma^{(n)}(f)\bigg|_{a_1=\cdots=a_n=0,\delta=\delta_0} = \prod_{i=1}^{n} A_i \sigma^{(n)}(\delta_0) \in \overline{\sigma(\Sigma_{c,1}^{l})}. \tag{2.96}$$

Also $\overline{\sigma(\Sigma_{c,1}^{l})}$ is a vector space, and $\prod_{i=1}^{n} A_i \in \overline{\sigma(\Sigma_{c,1}^{l})}$ which completes the proof. □

The lemma implies that the sufficiently smooth activation function can transform input functions to the componentwise product.

Now we provide the main proposition which shows that the minimum width $d-1$ is sufficient for the case of $d \geq 4$.

Proposition 2.7. *For the non-polynomial continuous activation function σ and $d \geq 4$, $(d-1)$-layered convolutional neural networks have the universal property in the continuous function space; that is, $\overline{\Sigma_{c,c'}^{d-1}(K)} = C(K, \mathbb{R}^{c'})$.*

Proof. Before we go any further, we denote that we only have to prove that $\overline{\Sigma_{c,1}^{d-1}(K)} = C(K, \mathbb{R})$ because the concatenation of the function can be conducted by Lemma 2.1. The flow of the proof follows the idea of [26]. The main idea is that if we can approximate all polynomials, all continuous functions in the compact domain can be approximated by the Stone–Weierstrass theorem [8]. The core difference is to make all multivariate polynomials in all positions of the output vector independently. The complexity made by convolution is the real matter that makes the problem tricky.

The proof is divided into the following steps. First, we will list the functions that can be approximated by convolution under the assumption that the activation function σ is a non-polynomial C^∞ function. Next, we construct the projection, which enables us to split each component of the output vector and construct an arbitrary polynomial in an arbitrary position. Finally, we generalize the result for the general non-polynomial activation function case later.

For the input vector $x = (x^1, x^2, \dots, x^c) \in \mathbb{R}^{c\times d}$, define the translation of $x^i = (x^i_1, x^i_2, \dots, x^i_d)$ as follows:

$$p^i_{-j} := U_j x^i = (0, \dots, 0, x^i_1, x^i_2, \dots, x^i_{d-j}), \tag{2.97}$$

$$p^i_0 := x^i = (x^i_1, x^i_2, \dots, x^i_d), \tag{2.98}$$

and

$$p^i_j := U_{-j} x^i = (x^i_{j+1}, \dots, x^i_{d-1}, x^i_d, 0, \dots, 0), \tag{2.99}$$

Case 1. $d = 4$**:** It is obvious by definition that $p^i_j = U_{-j}x^i \in \overline{\Sigma^1_{c,1}}$ for $j \in \{-1, 0, 1\}$. By Lemma 2.6, an arbitrary product of p^i_j is in $\overline{\sigma(\Sigma^1_{c,1})}$. In other words, for some constants $\alpha_{i,j} \in \mathbb{N}$ for $i \in [1, c], j \in \{-1, 0, 1\}$,

$$\prod_{i=1}^{c}\prod_{j=-1}^{1} (p^i_j)^{\alpha_{i,j}} \in \overline{\sigma(\Sigma^1_{c,1})}. \tag{2.100}$$

Consider vector-valued functions $A^1, A^2, \dots, A^n \in \overline{\sigma(\Sigma^1_{c,1})}$, and $\mathbf{1}_4 \in \overline{\sigma(\Sigma^1_{c,1})}$. Also, consider convolutional layers with kernel $b^i = (b^i_{-1}, b^i_0, b^i_1)$. By Lemma 2.1,

$$b^i \circledast A^i \in \overline{\Sigma^2_{c,1}}, \tag{2.101}$$

for $i \in [1, n]$, and

$$b^{n+1} \circledast \mathbf{1}_4 \in \overline{\Sigma^2_{c,1}}. \tag{2.102}$$

We construct the second convolutional layer B with n input channel and one output channel, which consists of convolutions with kernel and the bias δ. By Lemma 2.6, the Hadamard product of $b^i \circledast A^i$ is in $\overline{\sigma(\Sigma_{c,1}^2)}$:

$$\prod_{i=1}^{n} \left(b^i \circledast A^i\right) \in \overline{\sigma(\Sigma_{c,1}^2)}. \tag{2.103}$$

Now we construct the projection of the vectors $\prod_{i=1}^{n} \left(b^i \circledast A^i\right)$ to a certain axis. Because $b^{n+1} \circledast \mathbf{1}_4 \in \overline{\Sigma_{c,1}^2}$ and $\mathbf{1}_4 \in \overline{\Sigma_{c,1}^2}$, the linear summation of two functions is also in $\overline{\Sigma_{c,1}^2}$:

$$b^{n+1} \circledast \mathbf{1}_4 + \delta \mathbf{1}_4 \in \overline{\Sigma_{c,1}^2}. \tag{2.104}$$

Componentwise expression becomes

$$\begin{aligned} b^{n+1} \circledast \mathbf{1}_4 + \delta \mathbf{1}_4 &= \delta \mathbf{1}_4 + \\ &(b_0^{n+1} + b_1^{n+1}, b_{-1}^{n+1} + b_0^{n+1} + b_1^{n+1}, b_{-1}^{n+1} + b_0^{n+1} + b_1^{n+1}, b_{-1}^{n+1} + b_0^{n+1}). \end{aligned} \tag{2.105}$$

With $\delta = -(b_{-1}^{n+1} + b_0^{n+1} + b_1^{n+1})$, $b^{n+1} \circledast \mathbf{1}_4 + \delta \mathbf{1}_4$ becomes

$$b^{n+1} \circledast \mathbf{1}_4 + \delta \mathbf{1}_4 = (-b_{-1}^{n+1}, 0, 0, -b_1^{n+1}). \tag{2.106}$$

Therefore, $e_1 = (1, 0, 0, 0)$ and $e_4 = (0, 0, 0, 1)$ are in $\overline{\Sigma_{c,1}^2}$. By Lemma 2.6, the Hadamard product of $b^i \circledast A^i$ and e_1 is in $\overline{\sigma(\Sigma_{c,1}^2)}$:

$$\mathrm{pr}_1 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i\right) \right) = e_1 \odot \left(\prod_{i=1}^{n} \left(b^i \circledast A^i\right) \right) \in \overline{\sigma(\Sigma_{c,1}^2)}, \tag{2.107}$$

where pr_i means the projection to the i-th axis; that is, $\mathrm{pr}_1(\theta_1, \theta_2, \theta_3, \theta_4) = (\theta_1, 0, 0, 0)$, and $\mathrm{pr}_4(\theta_1, \theta_2, \theta_3, \theta_4) = (0, 0, 0, \theta_4)$. Similarly, the Hadamard product of $b^i \circledast A^i$ and e_4 is in $\overline{\sigma(\Sigma^2_{c,1})}$:

$$\mathrm{pr}_4 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right) = e_4 \odot \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right) \in \overline{\sigma(\Sigma^2_{c,1})}. \tag{2.108}$$

We also know that

$$\begin{aligned} &\mathrm{pr}_2 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right) + \mathrm{pr}_3 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right) \\ &\quad = \prod_{i=1}^{n} \left(b^i \circledast A^i \right) - \mathrm{pr}_1 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right) - \mathrm{pr}_4 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right). \end{aligned} \tag{2.109}$$

Therefore, $\mathrm{pr}_2 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right) + \mathrm{pr}_3 \left(\prod_{i=1}^{n} \left(b^i \circledast A^i \right) \right) \in \overline{\sigma(\Sigma^2_{c,1})}$.

Now, we construct the desired polynomials using the ingredients made in the previous steps. First, we will prove that for an monomial M_1 consisting of x^i_1, x^i_2, x^i_3, except x^i_4, $(M_1, 0, 0, 0)$ is the element of $\overline{\sigma(\Sigma^2_{c,1})}$. More concretely, M_1 is defined as

$$M_1 = \prod_{i=1}^{c} \prod_{j=1,2,3} (x^i_j)^{\alpha_{i,j}}, \tag{2.110}$$

where $\alpha_{i,j} \in \mathbb{N}_0$. Let A be

$$A = \prod_{i}^{c} \prod_{j=-1,0,1} (p^i_j)^{\alpha_{i,j+2}} \in \overline{\sigma(\Sigma^1_{c,1})}. \tag{2.111}$$

Then with $b = (0,0,1)$, $\mathrm{pr}_1 (b \circledast A) = \mathrm{pr}_1 (U_{-1}A) \in \overline{\Sigma^2_{c,1}}$, which means

$$\begin{aligned}
&\mathrm{pr}_1(U_{-1}A) && (2.112)\\
&= \mathrm{pr}_1 \left(\prod_{i=1}^{c} \prod_{j=1,2,3} (x^i_j)^{\alpha_{i,j}}, \prod_{i=1}^{c} \prod_{j=1,2,3} (x^i_{j+1})^{\alpha_{i,j}}, \prod_{i=1}^{c} \prod_{j=1,2,3} (x^i_{j+2})^{\alpha_{i,j}}, 0 \right) && (2.113)\\
&= \left(\prod_{i=1}^{c} \prod_{j=1,2,3} (x^i_j)^{\alpha_{i,j}}, 0, 0, 0 \right) \in \overline{\sigma(\Sigma^2_{c,1})}, && (2.114)
\end{aligned}$$

where $x^i_5 := 0$. Similarly, for a monomial M_2 consisting of x^i_2, x^i_3, x^i_4, except x^i_1, $(0,0,0,M_2)$ is the element of $\overline{\sigma(\Sigma^2_{c,1})}$; that is, for $\alpha_{i,j} \in \mathbb{N}_0$,

$$M_2 = \prod_{i=1}^{c} \prod_{j=2,3,4} (x^i_j)^{\alpha_{i,j}}. \tag{2.115}$$

The proof is obvious from symmetry.

Next, we will prove that for a monomial M_3 that contains at least one x^i_4, $(0, M_3, 0, 0)$ is the element of $\overline{\sigma(\Sigma^2_{c,1})}$; that is, for M_3 defined as

$$M_3 = x^{i_0}_4 \prod_{i=1}^{c} \prod_{j=1,2,3,4} (x^i_j)^{\alpha_{i,j}}, \tag{2.116}$$

$(0, M_3, 0, 0) \in \overline{\sigma(\Sigma^2_{c,1})}$ where $\alpha_{i,j} \in \mathbb{N}_0$. For the proof, define A_1 and A_2 as

$$A_1 = \prod_{i=1}^{c} \prod_{j=-1,0} (p^i_j)^{\alpha_{i,j+2}} \in \overline{\sigma(\Sigma^1_{c,1})}, \tag{2.117}$$

and

$$A_2 = p^{i_0}_1 \odot \prod_{i=1}^{c} \prod_{j=1,2} (p^i_{j-1})^{\alpha_{i,j+2}} \in \overline{\sigma(\Sigma^1_{c,1})}. \tag{2.118}$$

Also, define B as follows:

$$B := (0,0,1) \circledast A_2 = U_{-1}A_2 \in \overline{\Sigma^2_{c,1}}. \tag{2.119}$$

Then, we have

$$(\mathrm{pr}_2 + \mathrm{pr}_3)(A_1 \odot B) \in \overline{\sigma(\Sigma^2_{c,1})}. \tag{2.120}$$

Because

$$(\mathrm{pr}_2 + \mathrm{pr}_3)(A_1 \odot B) = (\mathrm{pr}_2 + \mathrm{pr}_3)(A_1) \odot (\mathrm{pr}_2 + \mathrm{pr}_3)(B), \tag{2.121}$$

and

$$(\mathrm{pr}_2 + \mathrm{pr}_3)(B) = \left(0, x_4^{i_0} \prod_{i=1}^{c} \prod_{j=3,4} (x_j^i)^{\alpha_{i,j}}, 0, 0\right), \tag{2.122}$$

$(\mathrm{pr}_2 + \mathrm{pr}_3)(A_1 \odot B)$ becomes

$$(\mathrm{pr}_2 + \mathrm{pr}_3)(A_1 \odot B) = (\mathrm{pr}_2)(A_1) \odot (\mathrm{pr}_2)(B) \tag{2.123}$$

$$= \left(0, \prod_{i=1}^{c} \prod_{j=1,2} (x_j^i)^{\alpha_{i,j}}, 0, 0\right) \odot \left(0, x_4^{i_0} \prod_{i=1}^{c} \prod_{j=3,4} (x_j^i)^{\alpha_{i,j}}, 0, 0\right) \tag{2.124}$$

$$= \left(0, x_4^{i_0} \prod_{i=1}^{c} \prod_{j=1}^{4} (x_j^i)^{\alpha_{i,j}}, 0, 0\right) = (0, M_3, 0, 0) \in \overline{\sigma(\Sigma^2_{c,1})}. \tag{2.125}$$

Similarly, the symmetrical argument shows that for a monomial M_4 containing at least one x_1^i, $(0,0,M_4,0)$ is the element of $\overline{\sigma(\Sigma^2_{c,1})}$. What we have proven in this step is that

- for a monomial M_1 that does not contain any x_4^i, $(M_1, 0, 0, 0) \in \overline{\sigma(\Sigma^2_{c,1})}$,
- for a monomial M_2 that does not contain any x_1^i, $(0, 0, 0, M_2) \in \overline{\sigma(\Sigma^2_{c,1})}$,

- for a monomial M_3 that contains at least one x_4^i, $(0, M_3, 0, 0) \in \overline{\sigma(\Sigma_{c,1}^2)}$,

- and for a monomial M_4 that contains at least one x_1^i, $(0, 0, M_4, 0) \in \overline{\sigma(\Sigma_{c,1}^2)}$.

Now we will prove that for arbitrary monomial M_0, $(M_0, 0, 0, 0)$, $(0, M_0, 0, 0)$, $(0, 0, M_0, 0)$, and $(0, 0, 0, M_0)$ are in $\overline{\Sigma_{c,1}^3}$. By Lemma 2.1, for an arbitrary convolutional layer C and the function $f \in \overline{\sigma(\Sigma_{c,1}^2)}$, $C(f) \in \overline{\Sigma_{c,1}^3}$. If a monomial M contains at least one x_4^i for some $i \in [1, c]$, $(0, M, 0, 0) \in \overline{\sigma(\Sigma_{c,1}^2)}$. And for $C(x) = U_0 x$, $C((0, M, 0, 0)) = (0, M, 0, 0) \in \overline{\Sigma_{c,1}^3}$, and for $C(x) = U_{-1} x$, $C((0, M, 0, 0)) = (M, 0, 0, 0) \in \overline{\Sigma_{c,1}^3}$. Otherwise, if a monomial M does not contain any x_4^i, $(M, 0, 0, 0) \in \overline{\sigma(\Sigma_{c,1}^2)}$. And for $C(x) = U_0 x$, $C((M, 0, 0, 0)) = (M, 0, 0, 0) \in \overline{\Sigma_{c,1}^3}$, and for $C(x) = U_1 x$, $C((M, 0, 0, 0)) = (0, M, 0, 0) \in \overline{\Sigma_{c,1}^3}$. So for an arbitrary monomial M, $(M, 0, 0, 0)$ and $(0, M, 0, 0)$ are the elements of $\overline{\Sigma_{c,1}^3}$. And by symmetry, $(0, 0, M, 0)$ and $(0, 0, 0, M)$ are also in $\overline{\Sigma_{c,1}^3}$. It completes the proof for the case of $d = 4$.

Case 2. $d \geq 5$: The proof proceeds almost the same to Case 1. The difference is that unlike Case 1, we can construct all the projections pr_k for all $k \in [1, d]$ when $d \geq 5$. More concretely, for functions $A^i \in \overline{\sigma(\Sigma_{c,1}^{d-3})}$, q_i, kernels $b^i \in \mathbb{R}^3$, and q^i defined as

$$q^i := b^i \circledast A^i, \tag{2.126}$$

the following relation holds for all $k \in [1, d]$,

$$\mathrm{pr}_k \left(\prod_{i=1}^{n} q_i \right) \in \overline{\sigma(\Sigma_{c,1}^{d-2})}. \tag{2.127}$$

The proof is from the following steps.

Step 1. In this step, we will show that we can assign different constants to each axis. Let e_i be the i-th standard basis in Euclidean space. And define the constant function $\boldsymbol{e}_i : \mathbb{R}^d \to \mathbb{R}^d$ that has constant value e_i: $\boldsymbol{e}_i(x) = e_i$ for all $x \in \mathbb{R}^d$. Then what we will prove is that $\overline{\Sigma^{d-2}_{c,1}}$ contains $\boldsymbol{e}_i$ for $i \in [1, d-3] \bigcup [4, d]$. More generally, $\boldsymbol{e}_i \in \overline{\Sigma^n_{c,1}}$ for $i \in [1, n-1] \bigcup [d-n+2, d]$. It can be proved by the following mathematical induction.

1. For the case of $n = 2$, constant function $A(x) := \delta_1 \mathbf{1}_d \in \overline{\sigma(\Sigma^1_{c,1})}$. Then, for the convolutional layer B with kernel $b = (b_{-1}, b_0, b_1)$ and the bias δ_2,

$$B \circ A \in \overline{\Sigma^2_{c,1}}. \tag{2.128}$$

More specifically,

$$B \circ A = \delta_1 \mathbf{1}_d + (\delta_2(b_0 + b_1), \delta_2(b_{-1} + b_0 + b_1), \ldots, \delta_2(b_{-1} + b_0 + b_1), \delta_2(b_{-1} + b_0)). \tag{2.129}$$

Then, by substituting δ_1 for $\delta' - \delta_2(b_{-1} + b_0 + b_1)$, we get

$$B \circ A = \delta' \mathbf{1}_d + (-\delta_2 b_{-1}, 0, \ldots, 0, -\delta_2 b_1) \in \overline{\Sigma^2_{c,1}}. \tag{2.130}$$

for arbitrary b_{-1} and b_1. So $\boldsymbol{e}_1, \boldsymbol{e}_d \in \overline{\Sigma^2_{c,1}}$, and the induction hypothesis is satisfied for the case of $n = 2$.

2. Assume that for $n = n_0$, the induction hypothesis is satisfied, i.e., $\boldsymbol{e}_i \in \overline{\Sigma^{n_0}_{c,1}} \subset \overline{\sigma(\Sigma^{n_0}_{c,1})}$ for $i \in [1, n_0 - 1] \bigcup [d - n_0 + 2, d]$. Then, for the convolutional layer $C(x) := U_1 x$,

$$C \circ \boldsymbol{e}_{n_0 - 1} = \boldsymbol{e}_{n_0} \in \overline{\Sigma^{n_0+1}_{c,1}}. \tag{2.131}$$

Similarly, for the convolutional layer $C(x) = U_{-1}x$,

$$C \circ \boldsymbol{e}_{d-n_0+2} = \boldsymbol{e}_{d-n_0+1} \in \overline{\Sigma_{c,1}^{n_0+1}}. \tag{2.132}$$

Therefore, the induction hypothesis is satisfied for $n = n_0 + 1$, and $\overline{\Sigma_{c,1}^{d-3}}$ contains $\boldsymbol{e}_i$ for $i \in [1, d-4] \bigcup [5, d]$.

Step 2. In this step, we will similarly construct a polynomial to the case of $d = 4$ and show that its projection can also be constructed. We first prove that for the function $f : \mathbb{R}^{c \times d} \to \mathbb{R}^d$ defined as $f_j^i(x) = U_j x^i$, $f_j^i \in \overline{\Sigma_{c,1}^l}$ for $j \in [-l, l]$. We use the mathematical induction. When $l = 1$, it is obviously satisfied. Assume that the induction hypothesis is satisfied for l: $f_j^i \in \overline{\Sigma_{c,1}^l}$ for $j \in [-l, l]$. By Lemma 2.6, $f_j^i \in \overline{\sigma(\Sigma_{c,1}^l)}$, and for $C(x) = U_1(x)$, $C \circ f_j^i \in \overline{\Sigma_{c,1}^{l+1}}$. And because $C \circ f_j^i = f_{j+1}^i$, for $j \in [-l, l]$, $C \circ f_{j+1}^i \in \overline{\Sigma_{c,1}^{l+1}}$. Similarly, using $C(x) = U_{-1}(x)$, we have $C \circ f_j^i = f_{j-1}^i$, for $j \in [-l, l]$. Therefore, the induction hypothesis is satisfied for $l + 1$.

Consider $l = d - 2$. Then $p_j^i = U_{-j} x^i \in \overline{\Sigma_{c,1}^{d-2}}$ for $j \in [-d+2, d-2]$. By Lemma 2.6, for $\alpha_{i,j} \in \mathbb{N}_0$, the following relation holds:

$$\prod_{i=1}^{c} \prod_{j=-d+2}^{d-2} (p_j^i)^{\alpha_{i,j}} \in \overline{\sigma(\Sigma_{c,1}^{d-2})}. \tag{2.133}$$

Additionally, consider $\boldsymbol{e}_t \in \overline{\sigma(\Sigma_{c,1}^{d-2})}$. Then by applying Lemma 2.6 to p_j^i and $\boldsymbol{e}_t$, we get

$$\boldsymbol{e}_t \odot \left(\prod_{i=1}^{c} \prod_{j=-d+2}^{d-2} (p_j^i)^{\alpha_{i,j}} \right) \in \overline{\sigma(\Sigma_{c,1}^{d-2})}. \tag{2.134}$$

Because for all $t \in [1, d-3] \bigcup [4, d]$, $\boldsymbol{e}_t$ is in $\overline{\Sigma_{c,1}^{d-2}}$, we are able to get the projection pr_t of $\prod_{i=1}^{c} \prod_{j=-d+2}^{d-2} (p_j^i)^{\alpha_{i,j}}$ for $t \in [1, d-3] \bigcup [4, d]$. For $d > 5$, $[1, d-3] \bigcup [4, d] = [1, d]$, so we have the projection to an arbitrary axis. For $d = 5$, $[1, d-3] \bigcup [4, d] = \{1, 2, 4, 5\}$, and because $\mathrm{pr}_3 = I_5 - \sum_{t=1,2,4,5} \mathrm{pr}_t$, the projection to an arbitrary axis is also available for $d = 5$.

Step 3. For an arbitrary monomial $M = \prod_{i=1}^{c} \prod_{j=1}^{d} (x_j^i)^{\alpha_{i,j}}$, we will show that the vector Me_t is in $\overline{\sigma(\Sigma_{c,1}^{d-2})}$:

$$Me_t = (0, \dots, 0, M, 0, \dots, 0) \in \overline{\sigma(\Sigma_{c,1}^{d-2})}, \tag{2.135}$$

for $t \in [2, d-1]$. We know that

$$\boldsymbol{e}_t \odot \left(\prod_{i=1}^{c} \prod_{j=-d+2}^{d-2} (p_j^i)^{\alpha_{i,j}} \right) \in \overline{\sigma(\Sigma_{c,1}^{d-2})}. \tag{2.136}$$

By proper calculation, we get

$$\begin{aligned} \boldsymbol{e}_t \odot \left(\prod_{i=1}^{c} \prod_{j=1}^{d} (p_{j+t}^i)^{\alpha_{i,j}} \right) = \prod_{i=1}^{c} \prod_{j=1}^{d} (e_t \odot p_{j+t}^i)^{\alpha_{i,j}} &= \prod_{i=1}^{c} \prod_{j=1}^{d} (x_j^i e_t)^{\alpha_{i,j}} & (2.137) \\ &= \prod_{i=1}^{c} \prod_{j=1}^{d} (x_j^i)^{\alpha_{i,j}} e_t = Me_t. & (2.138) \end{aligned}$$

Therefore, $Me_t \in \overline{\sigma(\Sigma_{c,1}^{d-2})}$ for $t \in [2, d-1]$.

Finally, by using proper U_1, U_0, U_{-1} for the last convolutional layer, we can get $Me_t \in \overline{\Sigma_{c,1}^{d-1}}$ for all $i \in [1, d]$, and it completes the proof for the non-polynomial C^∞ activation function σ.

Now remaining is to generalize the result of the non-polynomial C^∞ activation function for the general non-polynomial function. It comes from the Section 6 of the [26]. For any non-polynomial function σ, there exists the compact supported C^∞

function ϕ such that $\sigma * \phi$ is smooth and not a polynomial function(Step 5 and Step 6 of Section 6 [26]). And because $\sigma * \phi$ can be uniformly approximated by σ(Step 4 of Section 6 [26]), any convolutional neural network with the activation function $\sigma * \phi$ can be uniformly approximated by the convolutional neural networks with the activation function σ. And because CNN with the activation function $\sigma * \phi$ has the universal property, CNN with the activation function σ also has the universal property, and it completes the entire proof. □

Remark 2.8. *Translation equivariance is often referred to as the basis of the advantages of CNN models:*

$$f_s(x_t) = f_{s+i}(x_{t+i}). \tag{2.139}$$

In fact, infinite-length convolution without padding is translation equivariant. However, this property contradicts the universal property because of the relation between the output vector and the input vector. Actually, as shown in the proof process, padding plays an important role. The asymmetry that starts at the boundary gradually propagates toward the center, making it possible to achieve the universal property.

Lemma 2.9. *For the non-polynomial continuous activation function σ and $d = 2, 3$, d-layered convolutional neural networks has the universal property in the continuous function space; that is, $\overline{\Sigma^d_{c,c'}(K)} = C(K, \mathbb{R}^{c'})$.*

Proof. The proof is almost same to Proposition 2.7. Divide the case into $d = 2$ and $d = 3$.

Case 1 $d = 2$: For the vectors $p^i_{-1} = (0, x^i_1), p^i_0 = (x^i_1, x^i_2)$, and $, p^i_1 = (x^i_2, 0)$, Lemma 2.6 gives the following equation: for some constants $\alpha_{i,j} \in \mathbb{N}_0$ for $i \in$

$[1, c], j \in \{-1, 0, 1\}$,

$$\prod_{i,j} (p_j^i)^{\alpha_{i,j}} \in \overline{\sigma(\Sigma_{c,1}^1)}. \tag{2.140}$$

For a monomial M that contains at least one x_1^i, $(0, M)$ is the element of $\overline{\sigma(\Sigma_{c,1}^1)}$; that is, for $M = x_1^{i_0} \prod_{i=1}^c \prod_{j=1,2} (x_j^i)^{\alpha_{i,j}}$, $(0, M) \in \overline{\sigma(\Sigma_{c,1}^1)}$. it is obvious from the following equation.

$$p_{-1}^{i_0} \prod_{i,j} (p_j^i)^{\alpha_{i,j}} = (0, M) \in \overline{\sigma(\Sigma_{c,1}^1)}. \tag{2.141}$$

Then, for $C(x) = U_{-1}x$, $C((0, M)) = (M, 0) \in \overline{\Sigma_{c,1}^2}$, and for $C(x) = U_0 x$, $C((0, M)) = (0, M) \in \overline{\Sigma_{c,1}^2}$. By symmetric process, for a monomial M that contains at least one x_2^i, $(M, 0), (0, M) \in \overline{\Sigma_{c,1}^2}$. Now remaining is to prove that the constant functions $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are in $\overline{\Sigma_{c,1}^2}$. Because $(1, 1) \in \overline{\sigma(\Sigma_{c,1}^1)}$, $U_{-1}((1, 1)) = (1, 0) = \boldsymbol{e}_1 \in \overline{\Sigma_{c,1}^2}$, and $U_1((1, 1)) = (0, 1) = \boldsymbol{e}_2 \in \overline{\Sigma_{c,1}^2}$. It completes the proof for the case $d = 2$.

Case 2 $d = 3$: In the proof for Proposition 2.7, the following relation holds:

$$\boldsymbol{e}_i \in \overline{\Sigma_{c,1}^2}, \tag{2.142}$$

for $i \in [1, n-1] \bigcup [, d] = \{1, 3\}$. Because $p_j^i = U_{-j}x^i \in \overline{\Sigma_{c,1}^2}$ for $j \in [-2, 2]$, by Lemma 2.6, we have

$$\prod_{i=1}^{c} \prod_{j=-2}^{2} (p_j^i)^{\alpha_{i,j}} \in \overline{\sigma(\Sigma_{c,1}^2)}, \tag{2.143}$$

and

$$\boldsymbol{e}_t \odot \left(\prod_{i=1}^{c} \prod_{j=-2}^{2} (p_j^i)^{\alpha_{i,j}} \right) \in \overline{\sigma(\Sigma_{c,1}^2)}, \tag{2.144}$$

for $\alpha_{i,j} \in \mathbb{N}_0$ and $t \in \{1, 3\}$ Because $\mathrm{pr}_2 = I_3 - \mathrm{pr}_1 - \mathrm{pr}_3$, above equation is also satisfied for $t = 2$.

For an arbitrary monomial $M = \prod_{i=1}^{c} \prod_{j=1}^{3} (x_j^i)^{\alpha_{i,j}}$,

$$\mathrm{pr}_2 \left(\prod_{i}^{c} \prod_{j=1}^{3} (p_{j-2}^i)^{\alpha_{i,j}} \right) = (0, M, 0) \in \overline{\sigma(\Sigma_{c,1}^2)}. \tag{2.145}$$

Thus, using the convolutional layers U_{-1}, U_0, and U_1 as the last layer, $(M, 0, 0)$, $(0, M, 0)$,$(0, 0, M) \in \overline{\Sigma_{c,1}^3}$. And it completes the proof. □

Combining Lemma 2.9, Lemma 2.5, and Proposition 2.7 altogether, we get the following theorem:

Theorem 2.10. *For the non-polynomial continuous activation function σ, the minimal depth N_d for convolutional neural network to have the universal property is*

$$N_d = \begin{cases} 2 & \textit{if} \quad d = 1, 2, \\ 3 & \textit{else if} \quad d = 3, \\ d - 1 & \textit{else if} \quad d \geq 4. \end{cases} \tag{2.146}$$

In other words, for a compact set $K \subset \mathbb{R}^c$, $\overline{\Sigma_{c,c'}^{N_D}} = C(K, \mathbb{R}^{c'})$, and $\overline{\Sigma_{c,c'}^{N_D-1}} \neq C(K, \mathbb{R}^{c'})$.

2.2.4 The Minimum Width for the Universal Property of Convolutional Neural Network

In this section, we prove the universal property of deep narrow convolutional neural networks. The proof process is as follows. First, construct the convolutional neural networks, which can compute arbitrary linear summation of the input in Lemma 2.13. Second, in Lemma 2.14, compose the linear summation and the activation function to get the convolutional neural network which can approximate the arbitrary continuous function using only one activation function layer. Finally, construct the deep narrow neural network that can approximate the network mentioned above.

Lemma 2.11. *S_{d-1} contains the following elements.*

- *If $n+m \leq d-1$, $E_{n,m} \in S_{d-1}$.*
- *If $n+m \geq d+3$, $E_{n,m} \in S_{d-1}$.*
- *If $n+m = d+1$, $E_{n,m} \in S_{d-1}$.*
- *If $n+m = d$, $E_{n,m} + E_{n+1,m+1} \in S_{d-1}$.*

Proof.

- By simple operation, we can know that $U_0 - U_1U_{-1} = E_{1,1}$. And $U_1^{n-1}(U_0 - U_1U_{-1})U_{-1}^{m-1} = U_1^{n-1}U_{-1}^{m-1} - U_1^nU_{-1}^m = E_{n,m}$. So if $n+m \leq d-1$, $E_{n,m} \in S_{d-1}$.
- Similarly, $U_0 - U_{-1}U_1 = E(d,d)$. And $U_{-1}^{n-1}(U_0 - U_{-1}U_1)U_1^{m-1} = U_{-1}^{n-1}U_1^{m-1} - U_{-1}^nU_1^m = E(d-n+1, d-m+1)$. So if $(d-n+1)+(d-m+1) \geq d+3$, then $n+m \leq d-1$, and thus $E_{d-n+1,d-m+1} \in S_{d-1}$.
- Divide the case into two cases again. First, consider the case of $n \geq m$. Then, We can easily observe that $(U_1)^{n-m} = \sum_{i=-n+1}^{d-m} E_{n+i,m+i}$. Because $E_{n+i,m+i} \in S_{d-1}$ for all $i < 0(\because (n+i)+(m+i) = d+1+2i \leq d-1)$ and $i > 0(\because (n+i)+(m+i) = d+1+2i \geq d+3)$, and $(U_1)^{n-m} \in S_{d-1}$, $E_{n,m} = (U_1)^{n-m} - \sum_{i\neq 0} E_{n+i,m+i} \in S_{d-1}$. Similarly, if $n < m$, $(U_{-1})^{m-n} = \sum_{i=-n+1}^{m-1} E_{n+i,m+i}$, and thus $E_{n,m} = (U_{-1})^{m-n} - \sum_{i\neq 0} E_{n+i,m+i} \in S_{d-1}$.

- Similar to the above case, if $n \geq m$, then $(U_1)^{n-m} = \sum_{i=-n+1}^{d-m} E_{n+i,m+i}$. $E_{n,m} + E_{n+1,m+1} = (U_1)^{n-m} - \sum_{i \neq 0,1} E_{n+i,m+i} \in S_{d-1}$. If $n < m$, $E_{n,m} + E_{n+1,m+1} = (U_{-1})^{m-n} - \sum_{i \neq 0,1} E_{n+i,m+i} \in S_{d-1}$.

$\square$

Corollary 2.12. *For arbitrary $1 \leq n, m \leq d$, $E_{n,m} \in S_d$.*

Proof. Obviously, $S_{d-1} \subset S_d$. And $E_{n,m} \in S_d$, except for the cases of $n + m = d$ and $n + m = d + 2$. If $n + m = d$, $E_{n,m} = E_{n+1,m} U_1$. Because $n + 1 + m = d + 1$, $E_{n+1,m} \in S_{d-1}$, and thus $E_{n+1,m} U_1 \in S_d$. If $n + m = d + 2$, $E_{n,m} = E_{n-1,m} U_{-1}$. Because $n - 1 + m = d + 1$, $E_{n-1,m} \in S_{d-1}$, and thus $E_{n-1,m} U_{-1} \in S_d$. $\square$

Corollary 2.13. *For arbitrary matrix $L \in \mathbb{R}^{d \times d}$, $L \in S_d$.*

In the following lemma, we prove that the convolutional neural networks with only one activation function layer can approximate the arbitrary continuous function.

Lemma 2.14. *Define the set of functions as follows. For $x = (x^1, x^2, \ldots, x^c) \in \mathbb{R}^{c \times d}$ and $x^i \in \mathbb{R}^d$,*

$$T := \left\{ \sum_{j=1}^{n} a_j \sigma \left(\sum_{i=1}^{c} L_{j,i} x^i + \boldsymbol{\delta}_j \right) \middle| L_{j,i} \in \mathbb{R}^{d \times d}, \boldsymbol{\delta}_j \in \mathbb{R}^d, a_j \in \mathbb{R} \right\}, \tag{2.147}$$

where σ is the non-polynomial continuous activation function. Then, $\overline{T} = C(K, \mathbb{R}^{1 \times d})$ for the compact set $K \in \mathbb{R}^{c \times d}$.

Proof. Let x^i be $x^i = (x_1^i, x_2^i, \ldots, x_d^i) \in \mathbb{R}^d$. Define the arbitrary monomial of x_j^i as follows:

$$M = \prod_{i=1}^{c} \prod_{j=1}^{d} (x_j^i)^{\alpha_{i,j}}, \tag{2.148}$$

for some degrees $\alpha_{i,j} \in \mathbb{R}$. We will show that for $k \in [1, d]$,

$$Me_k = (0, 0, \dots, 0, M, 0, \dots, 0) \in \overline{T}. \tag{2.149}$$

Then, it is sufficient by Stone–Weierstrass theorem [8]. As in Lemma 2.1, $\overline{T}$, the closure of T, is a vector space and is closed under partial differentiation with respect to the parameters. For $\boldsymbol{\delta} = (\delta_1, \delta_2, \dots, \delta_d)$ and $b_{i,t} \in \mathbb{R}$,

$$\sigma \circ f(x) = \sigma \left(\sum_{i=1}^{c} b_{i,j} E_{k,j} x^i + \boldsymbol{\delta} \right) \in \overline{T}. \tag{2.150}$$

Then, partial differentiation with respect to δ_j and $b_{i,t}$ gives the following equation.

$$\left(\frac{\partial}{\partial \delta_k} \prod_{i=1}^{c} \prod_{j=1}^{d} \left(\frac{\partial}{\partial b_{i,j}} \right)^{\alpha_{i,j}} \right) \sigma(f) = \left(\prod_{i=1}^{c} \prod_{j=1}^{d} (x_j^i)^{\alpha_{i,j}} \right) e_k \odot \sigma^{(n)}(f), \tag{2.151}$$

where $n = \sum_{i=1}^{c} \sum_{j=1}^{d} \alpha_{i,j} + 1$. Then, with δ_j such that $\sigma^{(n)}(\delta_j) \neq 0$ and $b_{i,j} = 0$, we get,

$$Me_k \in \overline{T}. \tag{2.152}$$

Therefore, all polynomials are in $\overline{T}$, and by Stone–Weierstrass theorem, $\overline{T} = C(K, \mathbb{R}^{1 \times d})$. □

We demonstrate the universality of the deep narrow convolutional neural network in the next theorem.

Theorem 2.15. *Any function $f : \mathbb{R}^{c_x \times d} \to \mathbb{R}^{c_y \times d}$ can be approximated by convolutional neural networks with at most $c_x + c_y + 2$ channels and the non-polynomial continuous activation function; for any $\epsilon > 0$, there exists convolutional neural* *network g with $c_x + c_y + 2$ channels such that,*

$$||f - g||_{C^\infty(K)} < \epsilon. \tag{2.153}$$

Proof. First, consider the function f with c input channels and one output channel:

$$f : \mathbb{R}^{c\times d} \to \mathbb{R}^{1\times d}. \tag{2.154}$$

We denote the input as x and each channel of input as $x = (x^1, x^2, \ldots, x^c)$. By Lemma 2.14, there exist $g : \mathbb{R}^{c\times d} \to \mathbb{R}^{1\times d}$ such that defined as follows:

$$g(x) := \sum_{j=1}^{n} a_j \sigma \left(\sum_{i=1}^{c} L_{j,i} x^i + \boldsymbol{\delta}_j \right), \tag{2.155}$$

which can approximate f with an arbitrarily small error. Now construct the convolutional neural network with channel size $c + 3$ which approximates g. By Lemma 2.12, for arbitrary $L_{j,i} \in \mathbb{R}^{d\times d}$, there exists $C_{i,j}^{k,l} \in To_d(1)$ such that

$$L_{j,i} = \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l}. \tag{2.156}$$

Also, there exist $\widetilde{C}_j^{k,l} \in To_d(1)$ such that

$$\boldsymbol{\delta}_j = \sum_{l=1}^{\widetilde{m}_j} \prod_{k=1}^{d} \widetilde{C}_j^{k,l} \mathbf{1}_d. \tag{2.157}$$

Then, g becomes

$$g(x) = \sum_{j=1}^{n} a_j \sigma \left(\sum_{i=1}^{c} L_{j,i} + \boldsymbol{\delta}_j x^i \right)$$
$$= \sum_{j=1}^{n} a_j \sigma \left(\sum_{i=1}^{c} \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l} x^i + \sum_{l=1}^{\widetilde{m}_j} \prod_{k=1}^{d} \widetilde{C}_j^{k,l} \mathbf{1}_d \right). \quad (2.158)$$

Then we define the convolutional neural network with $c+3$ channels that calculate the aforementioned equation. By Lemma 2.2, if we can approximate the function with the convolutional neural network with the partial activation function, we can approximate the function with the original convolutional neural network. Therefore, we can preserve c channels from the input and process the $(c+1)$-th, $(c+2)$-th, and $(c+3)$-th channels. We get the desired output according to the following process of function compositions.

1. Repeat the following for $j = 1, 2, \dots, n$.

2. Calculate $\sigma \left(\sum_{i=1}^{c} \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l} x^i + \boldsymbol{\delta}_j \right)$ in the $(c+2)$-th channel, not using the $(c+3)$-th channel.

   2.1. Repeat the following for $i = 1, 2, \dots, c$ and $l = 1, 2, \dots, m_{i,j}$.

   2.2. Calculate $\prod_{k=1}^{d} C_{i,j}^{k,l} x^i$ in the $(c+1)$-th channel, not using the $(c+2)$-th and the $(c+3)$-th channels.

      2.2.1. Copy x^i from the i-th channel to the $(c+1)$-th channel.

      2.2.2. Conduct convolution with kernel $C_{i,j}^{k,l}$ and the bias 0 on the $(c+1)$-th channel for $k = 1, 2, \dots, d$.

   2.3. Add $\prod_{k=1}^{d} C_{i,j}^{k,l} x^i$ to the $(c+2)$-th channel and set the $(c+1)$-th channel to 0.

2.4. Add $\boldsymbol{\delta}_j = \sum_{l=1}^{\widetilde{m}_j} \prod_{k=1}^{d} \widetilde{C}_j^{k,l} \mathbf{1}_d$ to the $(c+2)$-th channel.

2.4.1. Repeat the following for $l = 1, 2, \ldots, \widetilde{m}_j$.

2.4.2. Conduct the convolution with kernel $(0, 0, 0)$ and the bias 1 on the $(c+1)$-th channel and get $\mathbf{1}_d$ on the $(c+1)$-th channel.

2.4.3. Conduct the convolution with kernel $\widetilde{C}_j^{k,l}$ and the bias 0 on the $(c+1)$-th channel for $k = 1, 2, \ldots, d$ and get $\prod_{k=1}^{d} \widetilde{C}_j^{k,l} \mathbf{1}_d$ in the $(c+1)$-th channel.

2.4.4. Add $\prod_{k=1}^{d} \widetilde{C}_j^{k,l} \mathbf{1}_d$ to the $(c+2)$-th channel and set the $(c+1)$-th channel to 0.

2.5. Apply the activation function on the $(c+2)$-th channel and get $\sigma\left(\sum_{i=1}^{c} \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l} x^i + \boldsymbol{\delta}_j\right)$ in the $(c+2)$-th channel.

3. Add $\sigma\left(\sum_{i=1}^{c} \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l} x^i + \boldsymbol{\delta}_j\right)$ to the $(c+3)$-th channel and set the $(c+2)$-th channel to 0.

4. Get $\sum_{j=1}^{n} a_j \sigma\left(\sum_{i=1}^{c} \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l} x^i + \boldsymbol{\delta}_j\right)$ in the $(c+3)$-th channel.

5. Set the final convolutional layer with one output channel, which takes the value from the $(c+3)$-th channel.

In this process, the $(c+1)$-th channel is used to calculate the product $\prod_{k=1}^{d} C_{i,j}^{k,l}$. And the $(c{+}2)$-th channel is used to accumulate the summation $\sum_{i=1}^{c} \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l} x^i$ calculated in the $(c+1)$-th channel. The $(c+3)$-th channel is used to accumulate the final summation $\sum_{j=1}^{n} a_j \sigma\left(\sum_{i=1}^{c} \sum_{l=1}^{m_{i,j}} \prod_{k=1}^{d} C_{i,j}^{k,l} x^i + \boldsymbol{\delta}_j\right)$ after the activation function is applied to the $(c+2)$-th channel. For the general case, when the output channel size is c_y, we can repeat the above process while preserving the output components already processed, and using $c_x + c_y + 2$ channels is enough to generate c_y output vectors. It completes the proof. $\square$

Chapter 3

The Universality Property of Deep Recurrent Neural Network

3.1 Terminologies and Notations

This section introduces the definition of network architecture and the notation used throughout this chapter. d_x and d_y denote the dimension of input and output space, respectively. σ is an activation function unless otherwise stated. Sometimes, v indicates a vector with suitable dimensions.

First, we used square brackets, subscripts, and colon symbols to index a sequence of vectors. More precisely, for a given sequence of d_x-dimensional vectors $x : \mathbb{N} \to \mathbb{R}^{d_x}$, $x[t]_j$ or $x_j[t]$ denotes the j-th component of the t-th vector. The colon symbol : is used to denote a continuous index, such as $x[a : b] = (x[i])_{a \le i \le b}$ or $x[t]_{a:b} = (x[t]_a, x[t]_{a+1}, \ldots, x[t]_b)^T \in \mathbb{R}^{b-a+1}$. We call the sequential index t by time and each $x[t]$ a *token*.

Second, we define the token-wise linear maps $\mathcal{P} : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_s \times N}$ and $\mathcal{Q} : \mathbb{R}^{d_s \times N} \to \mathbb{R}^{d_y \times N}$ to connect the input, hidden state, and output space. As the dimension of the hidden state space $\mathbb{R}^{d_s}$ on which the RNN cells act is different from those of the input domain $\mathbb{R}^{d_x}$ and output domain $\mathbb{R}^{d_y}$, we need maps adjusting the dimensions of the spaces. For a given matrix $P \in \mathbb{R}^{d_s \times d_x}$, a *lifting map* $\mathcal{P}(x)[t] \coloneqq Px[t]$ lifts the input vector to the hidden state space. Similarly, for a given matrix $Q \in \mathbb{R}^{d_y \times d_s}$, a *projection map* $\mathcal{Q}(s)[t] \coloneqq Qs[t]$ projects a hidden state onto the output vector. As the first token defines a token-wise map, we sometimes represent token-wise maps without a time length, such as $\mathcal{P} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_s}$ instead of $\mathcal{P} : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_s}$.

Subsequently, an RNN is constructed using a composition of basic recurrent cells between the lifting and projection maps. We considered four basic cells: RNN, LSTM, GRU, and BRNN.

- **RNN Cell** A *recurrent cell*, *recurrent layer*, or *RNN cell* $\mathcal{R}$ maps an input sequence $x = (x[1], x[2], \ldots) = (x[t])_{t \in \mathbb{N}} \in \mathbb{R}^{d_s \times \mathbb{N}}$ to an output sequence $y = (y[t])_{t \in \mathbb{N}} \in \mathbb{R}^{d_s \times \mathbb{N}}$ using

$$y[t+1] = \mathcal{R}(x)[t+1] = \sigma\left(A\mathcal{R}(x)[t] + Bx[t+1] + \theta\right), \tag{3.1}$$

  where σ is an activation function, $A, B \in \mathbb{R}^{d_s \times d_s}$ are the weight matrices, and $\theta \in \mathbb{R}^{d_s}$ is the bias vector. The initial state $y[0]$ can be an arbitrary constant vector, which is zero vector $\mathbf{0}$ in this setting.

- **LSTM cell** Mathematically, an *LSTM cell* $\mathcal{R}_{LSTM}$ is a process that computes two outputs, h and c, defined by the following relation:

$$
\begin{aligned}
f[t+1] &= \sigma_{\text{sig}}\left(W_f x[t+1] + U_f h[t] + V_f c[t] + b_f\right), \\
i[t+1] &= \sigma_{\text{sig}}\left(W_i x[t+1] + U_i h[t] + V_i c[t] + b_i\right), \\
\tilde{c}[t+1] &= \tanh\left(W_c x[t+1] + U_c h[t] + b_c\right), \\
c[t+1] &= f[t+1]c[t] + i[t+1]\tilde{c}[t+1], \\
o[t+1] &= \sigma_{\text{sig}}\left(W_o x[t+1] + U_o h[t] + V_o c[t+1] + b_o\right), \\
h[t+1] &= o[t+1]\tanh\left(c[t+1]\right),
\end{aligned}
\tag{3.2}
$$

where W_*, U_* and V_* are weight matrices; b_* is the bias vector for each $* = f, i, c, o$; and σ_{sig} is the sigmoid activation function. The initial state is zero in this thesis.

- **GRU cell** A *GRU cell* $\mathcal{R}_{GRU}$ is a process that computes h defined by

$$
\begin{aligned}
r[t+1] &= \sigma_{\text{sig}}\left(W_r x[t+1] + U_r h[t] + b_r\right), \\
\tilde{h}[t+1] &= \tanh\left(W_h x[t+1] + U_h\left(r[t+1] \odot h[t]\right) + b_h\right), \\
z[t+1] &= \sigma_{\text{sig}}\left(W_z x[t+1] + U_z h[t] + b_z\right), \\
h[t+1] &= \left(1 - z[t+1]\right)h[t] + z[t+1]\tilde{h}[t+1],
\end{aligned}
\tag{3.3}
$$

where W_* and U_* are weight matrices, b_* is the bias vector for each $* = r, z, h$, and σ_{sig} is the sigmoid activation function. $\odot$ denotes component-wise multiplication, and we set the initial state to zero in this study.

- **BRNN cell** A *BRNN cell* $\mathcal{BR}$ consists of a pair of RNN cells and a token-wise linear map that follows the cells. An RNN cell $\mathcal{R}_1$ in the BRNN cell $\mathcal{BR}$ receives input from $x[1]$ to $x[N]$ and the other $\mathcal{R}_2$ receives input from

$x[N]$ to $x[1]$ in reverse order. Then, the linear map $\mathcal{L}$ in $\mathcal{BR}$ combines the two outputs from the RNN cells. Specifically, a BRNN cell $\mathcal{BR}$ is defined as follows:

$$\begin{aligned}
\mathcal{R}(x)[t+1] &\coloneqq \sigma\left(A\mathcal{R}_1(x)[t] + Bx[t+1] + \theta\right), \\
\bar{\mathcal{R}}(x)[t-1] &\coloneqq \sigma\left(\bar{A}\bar{\mathcal{R}}(x)[t] + \bar{B}x[t-1] + \bar{\theta}\right), \\
\mathcal{BR}(x)[t] &\coloneqq \mathcal{L}\left(\mathcal{R}(x)[t], \bar{\mathcal{R}}(x)[t]\right) \\
&\coloneqq W\mathcal{R}(x)[t] + \bar{W}\bar{\mathcal{R}}(x)[t].
\end{aligned} \tag{3.4}$$

where A, B, $\bar{A}$, $\bar{B}$, W, and $\bar{W}$ are weight matrices; θ and $\bar{\theta}$ are bias vectors.

- **Network architecture** An *RNN* $\mathcal{N}$ comprises a lifting map $\mathcal{P}$, projection map $\mathcal{Q}$, and L recurrent cells $\mathcal{R}_1, \ldots, \mathcal{R}_L$;

$$\mathcal{N} \coloneqq \mathcal{Q} \circ \mathcal{R}_L \circ \cdots \circ \mathcal{R}_1 \circ \mathcal{P}. \tag{3.5}$$

  We denote the network as a *stack RNN* or *deep RNN* when $L \geq 2$, and each output of the cell $\mathcal{R}_i$ as the i-th hidden state. d_s indicates the width of the network. If LSTM, GRU, or BRNN cells replace recurrent cells, the network is called an LSTM, a GRU, or a BRNN.

In addition to the type of cell, the activation function σ affects universality. We focus on the case of ReLU or tanh while also considering the general activation function satisfying the condition proposed by [22]. σ is a continuous non-polynomial function that is continuously differentiable at some z_0 with $\sigma'(z_0) \neq 0$. We refer to the condition as a *non-degenerate* condition and z_0 as a *non-degenerating point.*

Finally, the target class must be set as a subset of the sequence-to-sequence function space, from $\mathbb{R}^{d_x}$ to $\mathbb{R}^{d_y}$. Given an RNN $\mathcal{N}$, each token $y[t]$ of the output sequence $y = \mathcal{N}(x)$ depends only on $x[1:t] \coloneqq (x[1], x[2], \ldots, x[t])$ for the input sequence x. We define this property as *past dependency* and a function with this property as a *past-dependent* function. More precisely, if all the output tokens of a sequence-to-sequence function are given by $f[t]\,(x[1:t])$ for functions $f[t] : \mathbb{R}^{d_x \times t} \to \mathbb{R}^{d_y}$, we say that the function is past-dependent. Meanwhile, we must fix the finite length or terminal time $N < \infty$ of the input and output sequence. Without additional assumptions such as in [15], errors generally accumulate over time, making it impossible to approximate implicit dynamics up to infinite time regardless of past dependency. Therefore we set the target function class as a class of past-dependent sequence-to-sequence functions with sequence length N.

Remark 3.1. *On a compact domain and under bounded length, the continuity of $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$ implies that of each $f[t] : \mathbb{R}^{d_x \times t} \to \mathbb{R}^{d_y}$ and vice versa. In the case of the L^p norm with $1 \le p < \infty$, $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$ is L^p integrable if and only if $f[t]$ is L^p integrable for each t. In both cases, the sequence of functions $(f_n)_{n \in \mathbb{N}}$ converges to g if and only if $(f_n[t])_{n \in \mathbb{N}}$ converges to $g[t]$ for each t. Thus, we focus on approximating $f[t]$ for each t under the given conditions.*

Sometimes, only the last value $\mathcal{N}(x)[N]$ is required considering an RNN $\mathcal{N}$ as a sequence-to-vector function $\mathcal{N} : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y}$. We freely use the terminology RNN for sequence-to-sequence and sequence-to-vector functions because there is no confusion when the output domain is evident.

We have described all the concepts necessary to set a problem, but we end this section with an introduction to the concepts used in the proof of the main theorem. For the convenience of the proof, we slightly modify the activation σ to act only on some components, instead of all components. With activation σ and index set

$I \subseteq \mathbb{N}$, *the modified activation* σ_I is defined as

$$\sigma_I(s)_i = \begin{cases} \sigma(s_i) & \text{if } i \in I, \\ s_i & \text{otherwise.} \end{cases} \tag{3.6}$$

Using the modified activation function σ_I, the basic cells of the network are modified in (3.1). For example, a *modified recurrent cell* can be defined as

$$\begin{aligned} \mathcal{R}(x)[t+1]_i &= \sigma_I\left(A\mathcal{R}(x)[t] + Bx[t+1] + \theta\right)_i \\ &= \begin{cases} \sigma\left(A\mathcal{R}(x)[t] + Bx[t+1] + \theta\right)_i & \text{if } i \in I, \\ \left(A\mathcal{R}(x)[t] + Bx[t+1] + \theta\right)_i & \text{otherwise.} \end{cases} \end{aligned} \tag{3.7}$$

Similarly, *modified RNN, LSTM, GRU, or BRNN* is defined using modified cells in (3.1). This concept is similar to the enhanced neuron of [22] in that activation can be selectively applied, but is different in that activation can be applied to partial components.

As activation leads to the non-linearity of a network, modifying the activation can affect the minimum width of the network. Fortunately, the following lemma shows that the minimum width increases by at most one owing to the modification. We briefly introduce the ideas here, with a detailed proof provided in Section 3.6.

Lemma 3.2. *Let* $\bar{\mathcal{R}} : \mathbb{R}^{d\times N} \to \mathbb{R}^{d\times N}$ *be a modified RNN cell,* $\bar{\mathcal{Q}} : \mathbb{R}^d \to \mathbb{R}^d$*, and* $\bar{\mathcal{P}} : \mathbb{R}^d \to \mathbb{R}^d$ *be a token-wise linear projection and lifting map. Suppose that an activation function* σ *of* $\bar{\mathcal{R}}$ *is non-degenerate with a non-degenerating point* z_0*. Then for any compact subset* $K \subset \mathbb{R}^d$ *and* $\epsilon > 0$*, there exists RNN cells* $\mathcal{R}_1$*,* $\mathcal{R}_2 : \mathbb{R}^{(d+\beta(\sigma))\times N} \to \mathbb{R}^{(d+\beta(\sigma))\times N}$*, and a token-wise linear map* $\mathcal{P} : \mathbb{R}^d \to \mathbb{R}^{d+\beta(\sigma)}$*,*

$\mathcal{Q} : \mathbb{R}^{d+\beta(\sigma)} \to \mathbb{R}^d$ *such that*

$$\sup_{x \in K^N} \left\| \bar{\mathcal{Q}} \circ \bar{\mathcal{R}} \circ \bar{\mathcal{P}}(x) - \mathcal{Q} \circ \mathcal{R}_2 \circ \mathcal{R}_1 \circ \mathcal{P}(x) \right\| < \epsilon, \tag{3.8}$$

where

$$\beta(\sigma) = \begin{cases} 0 & \textit{if } z_0 = 0, \\ 1 & \textit{otherwise.} \end{cases} \tag{3.9}$$

Sketch of proof. The detailed proof is available in Section 3.6.1. We use the Taylor expansion of σ at z_0 to recover the value before activation. For the i-th component with $i \notin I$, choose a small $\delta > 0$ and linearly approximate $\sigma(z_0 + \delta z)$ as $\sigma(z_0) + \delta\sigma'(z_0)z$. An affine transform after the Taylor expansion recovers z. □

Remark 3.3. *Note that the additional width only serves to translate some components after activation to use the Taylor expansion at z_0. We can remove the additional node if the activation function is in the closure of the set,*

$$\left\{\sigma : \mathbb{R} \to \mathbb{R} \mid \sigma \textit{ is non-degenerating at } 0\right\}, \tag{3.10}$$

or use an affine projection map instead of a linear projection map.

The lemma implies that a modified RNN can be approximated by an RNN with at most one additional width. For a given modified RNN $\bar{\mathcal{Q}} \circ \bar{\mathcal{R}}_L \circ \cdots \circ \bar{\mathcal{R}}_1 \circ \mathcal{P}$ of width d and $\epsilon > 0$, we can find RNN $\mathcal{R}_1, \ldots, \mathcal{R}_{2L}$ and linear maps $\mathcal{P}_1, \ldots, \mathcal{P}_L$, $\mathcal{Q}_1, \ldots, \mathcal{Q}_L$ such that

$$\begin{aligned}\sup_{x \in K^N} \| &\bar{\mathcal{Q}} \circ \bar{\mathcal{R}}_L \circ \cdots \circ \bar{\mathcal{R}}_1 \circ \bar{\mathcal{P}}(x) \\ &- \left(\mathcal{Q}_L \mathcal{R}_{2L} \mathcal{R}_{2L-1} \mathcal{P}_L\right) \circ \cdots \circ \left(\mathcal{Q}_1 \mathcal{R}_2 \mathcal{R}_1 \mathcal{P}_1\right)(x)\| < \epsilon.\end{aligned} \tag{3.11}$$

The composition $\mathcal{R} \circ \mathcal{P}$ of an RNN cell $\mathcal{R}$ and token-wise linear map $\mathcal{P}$ can be substituted by another RNN cell $\mathcal{R}'$. More concretely, for $\mathcal{R}$ and $\mathcal{P}$ defined by

$$\mathcal{R}(x)[t+1] = \sigma\left(A\mathcal{R}(x)[t] + Bx[t+1] + \theta\right), \tag{3.12}$$

$$\mathcal{P}(x)[t] = P\left(x[t]\right), \tag{3.13}$$

$\mathcal{R} \circ \mathcal{P}$ defines an RNN cell $\mathcal{R}'$

$$\mathcal{R}'(x)[t+1] = \sigma\left(A\mathcal{R}'(x)[t] + BPx[t+1] + \theta\right). \tag{3.14}$$

Thus, $\mathcal{R}_{2l+1}\left(\mathcal{P}_{l+1}\mathcal{Q}_l\right)$ becomes a recurrent cell, and the composition,

$$\left(\mathcal{Q}_L\mathcal{R}_{2L}\mathcal{R}_{2L-1}\mathcal{P}_L\right) \circ \cdots \circ \left(\mathcal{Q}_1\mathcal{R}_2\mathcal{R}_1\mathcal{P}_1\right)(x), \tag{3.15}$$

defines a network of form (3.5).

3.2 Universal Approximation for Deep RNN in Continuous Function Space

This section introduces the universal approximation theorem of deep RNNs in continuous function space.

Theorem 3.4 (Universal approximation theorem of deep RNN 1)**.** *Let $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$ be a continuous past-dependent sequence-to-sequence function and σ be a non-degenerate activation function. Then, for any $\epsilon > 0$ and compact subset*

$K \subset \mathbb{R}^{d_x}$, there exists a deep RNN $\mathcal{N}$ of width $d_x + d_y + 2 + \alpha(\sigma)$ such that

$$\sup_{x \in K^N} \sup_{1 \le t \le N} \| f(x)[t] - \mathcal{N}(x)[t] \| < \epsilon, \tag{3.16}$$

where

$$\alpha(\sigma) = \begin{cases} 0 & \sigma \text{ is ReLU,} \\ 1 & \sigma \text{ is a non-degenerating function with } \sigma(z_0) = 0, \\ 2 & \sigma \text{ is a non-degenerating function with } \sigma(z_0) \neq 0. \end{cases} \tag{3.17}$$

To prove the above theorem, we deal with the case of the sequence-to-vector function $\mathcal{N} : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y}$ first. Then, we extend our idea to a sequence-to-sequence function using bias terms to separate the input vectors at different times.

The main concept of the proof consists of three steps. First, we embed the sequential input $x[1:t]$ into D_t for the disjoint subsets $D_1, \dots, D_N$ using bias and a recurrent process. By embedding, effect of $x[t]$ on $y[t]$ and that of $x[t+1]$ on $y[t+1]$ will be completely independent. Embedding is unnecessary in the sequence-to-vector case, where we consider only the last output $y[N]$. Next, we find a two-layered MLP approximating the given target function and construct a modified RNN in Lemma 3.5 that simulates the hidden node of the MLP. The node of the MLP calculates the linear sum of all Nd_x input components, which can be represented as the sum of the inner product of some matrices and N input vectors in $\mathbb{R}^{d_x}$. Finally, an additional buffer component of the modified RNN cell copies another hidden node in the two-layered MLP. Then, the following modified RNN cell accumulates two results from the copied nodes. The buffer component of the modified RNN cell is then reset to zero to copy another hidden node of the MLP.

As this procedure is repeated, the modified RNN with bounded width copies the two-layered MLP. As the number of additional components required in each step depends on the activation function, we use $\alpha(\sigma)$ to state the theorem briefly.

Now, we present the statements and sketches of the proof corresponding to each step. The following lemma implies that a modified RNN computes the linear sum of all the input components, which copies the hidden node of a two-layered MLP.

Lemma 3.5. *Suppose $A[1], A[2], \cdots, A[N] \in \mathbb{R}^{1\times d_x}$ are the given matrices. Then there exists a modified RNN $\mathcal{N} = \mathcal{R}_L \circ \mathcal{R}_{L-1} \circ \cdots \circ \mathcal{R}_1 \circ \mathcal{P} : \mathbb{R}^{d_x\times N} \to \mathbb{R}^{(d_x+1)\times N}$ of width d_x+1 such that (the symbol $*$ indicates that there exists some value irrelevant to the proof)*

$$\begin{aligned} \mathcal{N}(x)[t] &= \begin{bmatrix} x[t] \\ * \end{bmatrix} && \textit{for } t < N, \\ \mathcal{N}(x)[N] &= \begin{bmatrix} x[N] \\ \sigma\left(\sum_{t=1}^{N} A[t]x[t]\right) \end{bmatrix}. \end{aligned} \tag{3.18}$$

Sketch of the proof. The detailed proof is available in Section 3.6.2. Define the m-th modified RNN cell $\mathcal{R}_m$, of the form of (3.1) without activation, with $A_m = \begin{bmatrix} O_{d_x\times d_x} & O_{d_x\times 1} \\ O_{1\times d_x} & 1 \end{bmatrix}$, $B_m = \begin{bmatrix} I_{d_x} & O_{d_x\times 1} \\ b_m & 0 \end{bmatrix}$ where $b_m \in \mathbb{R}^{1\times b_x}$. Then, the (d_x+1)th component $y[N]_{d_x+1}$ of the final output $y[N]$ after N layers becomes a linear combination of $b_i x[j]$ with some constant coefficients $\alpha_{i,j}$ and $\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_{i,j} b_i x[j]$. Thus the coefficient of $x[j]$ is represented by $\sum_{i=1}^{N} \alpha_{i,j} b_i$, which we wish to be $A[j]$ for each $j = 1, 2, \ldots, N$. In matrix formulation, we intend to find b satisfying $\Lambda^T b = A$, where $\Lambda = \{\alpha_{i,j}\}_{1\le i,j\le N} \in \mathbb{R}^{N\times N}$, $b = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix} \in \mathbb{R}^{N\times d_x}$, and $A = \begin{bmatrix} A[1] \\ \vdots \\ A[N] \end{bmatrix}$. As Λ is invertible there exist b_i that solve $\left(\Lambda^T b\right)_j = A[j]$. □

After copying a hidden node using the above lemma, we add a component, (d_x+2)th, to copy another hidden node. Then the results are accumulated in the (d_x+1)th component, and the final component is to be reset to copy another node. As the process is repeated, a modified RNN replicates the output node of a two-layered MLP.

Lemma 3.6. *Suppose $w_i \in \mathbb{R}$, $A_i[t] \in \mathbb{R}^{1\times d_x}$ are given for $t = 1, 2, \ldots, N$ and $i = 1, 2, \ldots, M$. Then, there exists a modified RNN $\mathcal{N} : \mathbb{R}^{d_x\times N} \to \mathbb{R}$ of width d_x+2 such that*

$$\mathcal{N}(x) = \sum_{i=1}^{M} w_i \sigma\left(\sum_{t=1}^{N} A_i[t]x[t]\right). \tag{3.19}$$

Proof. First construct a modified RNN $\mathcal{N}_1 : \mathbb{R}^{d_x\times N} \to \mathbb{R}^{(d_x+2)\times N}$ of width d_x+2 such that

$$\mathcal{N}_1(x)[t] = \begin{bmatrix} x[t] \\ * \\ 0 \end{bmatrix} \qquad \text{for } t < N, \tag{3.20}$$

$$\mathcal{N}_1(x)[N] = \begin{bmatrix} x[N] \\ \sigma\left(\sum_{t=1}^{N} A_1[t]x[t]\right) \\ 0 \end{bmatrix}, \tag{3.21}$$

as Lemma 3.5. Note that the final component does not affect the first linear summation and remains zero. Next, using the components except for the (d_x+1)th one, construct $\mathcal{N}_2 : \mathbb{R}^{(d_x+2)\times N} \to \mathbb{R}^{(d_x+2)\times N}$, which satisfies

$$\mathcal{N}_2\mathcal{N}_1(x)[t] = \begin{bmatrix} x[t] \\ * \\ * \end{bmatrix} \qquad \text{for } t < N, \tag{3.22}$$

$$\mathcal{N}_2\mathcal{N}_1(x)[N] = \begin{bmatrix} x[N] \\ \sigma\left(\sum_{t=1}^{N} A_1[t]x[t]\right) \\ \sigma\left(\sum_{t=1}^{N} A_2[t]x[t]\right) \end{bmatrix}, \tag{3.23}$$

and use one modified RNN cell $\mathcal{R}$ after $\mathcal{N}_2$ to add the results and reset the last component:

$$\mathcal{R}\mathcal{N}_2\mathcal{N}_1(x)[t] = \begin{bmatrix} x[t] \\ * \\ 0 \end{bmatrix}, \tag{3.24}$$

$$\mathcal{R}\mathcal{N}_2\mathcal{N}_1(x)[N] = \begin{bmatrix} x[N] \\ w_1\sigma\left(\sum A_1[t]x[t]\right) + w_2\sigma\left(\sum A_2[t]x[t]\right) \\ 0 \end{bmatrix}. \tag{3.25}$$

As the (d_x+2)th component is reset to zero, we use it to compute the third sum $w_3\sigma\left(\sum A_3[t]x[t]\right)$ and repeat until we obtain the final network $\mathcal{N}$ such that

$$\mathcal{N}(x)[N] = \begin{bmatrix} x[N] \\ \sum_{i=1}^{M} w_i\sigma\left(\sum_{t=1}^{N} A_i[t]x[t]\right) \\ 0 \end{bmatrix}. \tag{3.26}$$

$\square$

Remark 3.7. *The above lemma implies that a modified RNN of width $d_x + 2$ can copy the output node of a two-layered MLP. We can extend this result to an arbitrary d_y-dimensional case. Note that the first d_x components remain fixed, the $(d_x + 1)$th component computes a part of the linear sum approximating the target function, and the (d_x+2)th component computes another part and is reset. When we need to copy another output node for another component of the output of the target function $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$, only one additional width is sufficient. Indeed, the (d_x+2)th component computes the sum and the final component, and the (d_x+3)th component acts as a buffer to be reset in that case. By repeating this process, we obtain (d_x+d_y+1)-dimensional output from the modified RNN, which includes all d_y outputs of the MLP and the components from the $(d_x + 1)$th to the $(d_x + d_y)$th ones.*

Theorem 3.8 (Universal approximation theorem of deep RNN 2)**.** *Suppose $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y}$ is a continuous sequence-to-vector function, $K \subset \mathbb{R}^{d_x}$ is a compact subset, σ is a non-degenerating activation function, and z_0 is the non-degenerating point. Then, for any $\epsilon > 0$, there exists a deep RNN $\mathcal{N} : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y}$ of width $d_x + d_y + 1 + \beta(\sigma)$ such that*

$$\sup_{x \in K^N} \|f(x) - \mathcal{N}(x)\| < \epsilon, \tag{3.27}$$

where

$$\beta(\sigma) = \begin{cases} 0 & \text{if } z_0 = 0, \\ 1 & \text{otherwise.} \end{cases} \tag{3.28}$$

Proof. We present the proof for $d_y = 1$ here, but adding $d_y - 1$ width for each output component works for the case $d_y > 1$. By the universal approximation theorem of the MLP, there exist w_i and $A_i[t]$ for $i = 1, \ldots, M$ such that

$$\sup_{x \in K^N} \left\| f(x) - \sum_{i=1}^{M} w_i \sigma \left(\sum_{t=1}^{N} A_i[t] x[t] \right) \right\| < \frac{\epsilon}{2}. \tag{3.29}$$

Note that there exists a modified RNN $\bar{\mathcal{N}} : \mathbb{R}^{d_x \times N} \to \mathbb{R}$ of width $d_x + 2$,

$$\bar{\mathcal{N}}(x) = \sum_{i=1}^{M} w_i \sigma \left(\sum_{t=1}^{N} A_i[t] x[t] \right). \tag{3.30}$$

By Lemma 3.2, there exists an RNN $\mathcal{N} : \mathbb{R}^{d_x \times N} \to \mathbb{R}$ of width $d_x + 2 + \beta(\sigma)$ such that

$$\sup_{x \in K^n} \left\| \bar{\mathcal{N}}(x) - \mathcal{N}(x) \right\| < \frac{\epsilon}{2}. \tag{3.31}$$

Hence we have $\|f(x) - \mathcal{N}(x)\| < \epsilon$. □

Now, we consider an RNN $\mathcal{R}$ as a function from sequence x to sequence $y = \mathcal{R}(x)$ defined by (3.1). Although the above results are remarkable in that the minimal width has an upper bound independent of the length of the sequence, it only approximates a part of the output sequence. Meanwhile, as the hidden states calculated in each RNN cell are connected closely for different times, fitting all the functions that can be independent of each other becomes a more challenging problem. For example, the coefficient of $x[t-1]$ in $\mathcal{N}(x)[t]$ equals the coefficient of $x[t]$ in $\mathcal{N}(x)[t+1]$ if $\mathcal{N}$ is an RNN defined as in the proof of Lemma 3.5. This correlation originates from the fact that $x[t-1]$ and $x[t]$ arrive at $\mathcal{N}(x)[t], \mathcal{N}(x)[t+1]$ via the same intermediate process, 1-time step, and N layers.

We sever the correlation between the coefficients of $x[t-1]$ and $x[t]$ by defining the *time-enhanced recurrent cell* as follows:

Definition 3.9. *Time-enhanced recurrent cell, or layer, is a process that maps sequence* $x = (x[t])_{t\in\mathbb{N}} \in \mathbb{R}^{d_s\times\mathbb{N}}$ *to sequence* $y = (y[t])_{t\in\mathbb{N}} \in \mathbb{R}^{d_s\times\mathbb{N}}$ *via*

$$y[t+1] := \mathcal{R}(x)[t+1] = \sigma\left(A[t+1]\mathcal{R}(x)[t] + B[t+1]x[t+1] + \theta[t+1]\right) \quad (3.32)$$

where σ *is an activation function,* $A[t]$, $B[t] \in \mathbb{R}^{d_s\times d_s}$ *are weight matrices and* $\theta[t] \in \mathbb{R}^{d_s}$ *is the bias given for each time step* t.

Like RNN, *time-enhanced RNN* indicates a composition of the form (3.1) with time-enhanced recurrent cells instead of RNN cells, and we denote it as TRNN. The *modified TRNN* indicates a TRNN whose activation functions in some cell act on only part of the components. *Time-enhanced BRNN*, denoted as *TBRNN*, indicates a BRNN whose recurrent layers in each direction are replaced by time-enhanced layers. A *modified TBRNN* indicates a TBRNN whose activation function is modified to act on only part of the components. With the proof of Lemma 3.2 using $\bar{A}[t]$, $\bar{B}[t]$ instead of $\bar{A}$, $\bar{B}$, a TRNN can approximate a modified TRNN.

The following lemma shows that the modified TRNN successfully eliminates the correlation between outputs. See the Section 3.6 for the complete proof.

Lemma 3.10. *Suppose* $A_j[t] \in \mathbb{R}^{1\times d_x}$ *are the given matrices for* $1 \le t \le N$, $1 \le j \le t$. *Then there exists a modified TRNN* $\tilde{\mathcal{N}} : \mathbb{R}^{d_x\times N} \to \mathbb{R}^{(d_x+1)\times N}$ *of width* $d_x + 1$ *such that*

$$\tilde{\mathcal{N}}(x)[t] = \begin{bmatrix} x[t] \\ \sigma\left(\sum_{j=1}^{t} A_j[t]x[j]\right) \end{bmatrix}, \quad (3.33)$$

for all $t = 1, 2, \ldots, N$.

Sketch of proof. The detailed proof is available in Section 3.6.3. Use $b_m[t]$ instead of b_m in the proof of Lemma 3.5. As the coefficient matrices at each time $[t]$ after N

layers are full rank, we can find $b_m[t]$ implementing the required linear combination for each time. $\square$

Recall the proof of Theorem 3.6. An additional width serves as a buffer to implement and accumulate linear sum in a node in an MLP. Similarly, we proceed with Lemma 3.10 instead of Lemma 3.5 to conclude that there exists a modified TRNN $\mathcal{N}$ of width $d_x + 2$ such that each $\mathcal{N}[t]$ reproduces an MLP approximating $f[t]$.

Lemma 3.11. *Suppose $w_i \in \mathbb{R}$, $A_{i,j}[t] \in \mathbb{R}^{1\times d_x}$ are thr given matrices for $1 \le t \le N$, $1 \le j \le t$, $1 \le i \le M$. Then, there exists a modified TRNN $\tilde{\mathcal{N}} : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{1\times N}$ of width $d_x + 2$ such that*

$$\tilde{\mathcal{N}}(x)[t] = \sum_{i=1}^{M} w_i \sigma\left(\sum_{j=1}^{t} A_{i,j}[t]x[j]\right). \tag{3.34}$$

Proof. We omit the detailed proof because it is almost the same as the proof of Lemma 3.6. The only difference is to use Lemma 3.10 instead of Lemma 3.5. $\square$

This implies that the modified TRNN can approximate any past-dependent sequence-to-sequence function.

Finally, we connect the TRNN and RNN. Although it is unclear whether a modified RNN can approximate an arbitrary modified TRNN, there exists a modified RNN that approximates the specific one described in Lemma 3.10.

Lemma 3.12. *Let $\tilde{\mathcal{N}}$ be a given modified TRNN that computes* (3.33) *and $K \subset \mathbb{R}^{d_x}$ be a compact set. Then ,for any $\epsilon > 0$ there exists a modified RNN $\mathcal{N}$ of width*

$d_x + 2 + \gamma(\sigma)$ such that

$$\sup_{x \in K^N} \left\| \tilde{\mathcal{N}}(x) - \mathcal{N}(x) \right\| < \epsilon, \tag{3.35}$$

where $\gamma(\mathrm{ReLU}) = 0$, $\gamma(\sigma) = 1$ for non-degenerating activation σ.

Sketch of proof. The detailed proof is available in Section 3.6.4. Without loss of generality, we can assume $K \subset \left[0, \frac{1}{2}\right]^{d_x}$ and construct the first cell as the output at time t to be $x[t] + t\mathbf{1}_{d_x}$. As N compact sets $K + t\mathbf{1}_{d_x}$ are disjoint, there exists an MLP of width $d_x + 1 + \gamma(\sigma)$ approximating $x[t] + t\mathbf{1}_{d_x} \to b[t]x[t]$ as a function from $\mathbb{R}^{d_x} \to \mathbb{R}$ [14, 22]. Indeed, we need to approximate $x[t] + t\mathbf{1}_{d_x} \to \begin{bmatrix} x[t] + t\mathbf{1}_{d_x} \\ b[t]x[t] \end{bmatrix}$ as a function from $\mathbb{R}^{d_x}$ to $\mathbb{R}^{d_x+1}$. Fortunately, the first d_x components preserve the original input data in the proof of **Proposition 4.2(Register Model)** in [22]. Thus an MLP of width $d_x + 1$ approximates $b[t]x[t]$ while preserving the $x + t\mathbf{1}_{d_x}$ terms. Note that a token-wise MLP is a special case of an RNN of the same width. Nonetheless, we need an additional width to keep the $(d_x + 1)$th component approximating $b[t]x[t]$. Using the token-wise MLP implemented by an RNN and additional buffer width, we construct a modified RNN of width $d_x + 2 + \gamma(\sigma)$ approximating the modified TRNN cell used in the proof of Lemma 3.10. □

Summarizing all the results, we have the universality of a deep RNN in a continuous function space.

Proof of Theorem 3.4. As mentioned in Remark 3.7, we can set $d_y = 1$ for notational convenience. By Lemma 3.11, there exists a modified TRNN $\tilde{\mathcal{N}}$ of width

$d_x + 2$ such that

$$\sup_{x \in K^n} \left\| f(x) - \tilde{\mathcal{N}}(x) \right\| < \frac{\epsilon}{3}. \tag{3.36}$$

As $\tilde{\mathcal{N}}$ is a composition of modified TRNN cells of width $d_x + 2$ satisfying (3.33), there exists a modified RNN $\bar{\mathcal{N}}$ of width $d_x + 3 + \gamma(\sigma)$ such that

$$\sup_{x \in K^n} \left\| \tilde{\mathcal{N}}(x) - \bar{\mathcal{N}}(x) \right\| < \frac{\epsilon}{3}. \tag{3.37}$$

Then, by Lemma 3.2, there exists an RNN $\mathcal{N}$ of width $d_x + 3 + \gamma(\sigma) + \beta(\sigma) = d_x + 3 + \alpha(\sigma)$ such that

$$\sup_{x \in K^n} \left\| \bar{\mathcal{N}}(x) - \mathcal{N}(x) \right\| < \frac{\epsilon}{3}. \tag{3.38}$$

The triangle inequality yields

$$\sup_{x \in K^n} \left\| f(x) - \mathcal{N}(x) \right\| < \epsilon. \tag{3.39}$$

$\square$

Remark 3.13. *The number of additional widths $\alpha(\sigma) = \beta(\sigma) + \gamma(\sigma)$ depends on the condition of the activation function σ. Here, $\gamma(\sigma)$ is required to find the token-wise MLP that approximates embedding from $\mathbb{R}^{d_x}$ to $\mathbb{R}^{d_x+1}$. If further studies determine a tighter upper bound of the minimum width of an MLP to have the universal property in a continuous function space, we can reduce or even remove $\alpha(\sigma)$ according to the result.*

There is still a wide gap between the lower bound d_x and upper bound $d_x + d_y + 3 + \alpha(\sigma)$ of the minimum width, and hence, we expect to be able to achieve universality with a narrower width. For example, if $N = 1$, an RNN is simply an MLP, and the RNN has universality without a node required to compute the effect of t. Therefore, apart from the result of the minimum width of an MLP, further studies are required to determine whether γ is essential for the case of $N \geq 2$.

3.3 Universal Approximation for Stack RNN in L^p Space

This section introduces the universal approximation theorem of a deep RNN in L^p function space for $1 \leq p < \infty$.

Theorem 3.14 (Universal approximation theorem of deep RNN 3)**.** *Let $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$ be a past-dependent sequence-to-sequence function in $L^p\left(\mathbb{R}^{d_x \times N}, \mathbb{R}^{d_y \times N}\right)$ for $1 \leq p < \infty$, and σ be a non-degenerate activation function with the non-degenerating point z_0. Then, for any $\epsilon > 0$ and compact subset $K \subset \mathbb{R}^{d_x}$, there exists a deep RNN $\mathcal{N}$ of width* $\max\{d_x + 1, d_y\} + \gamma(\sigma)$ *satisfying*

$$\sup_{1 \leq t \leq N} \|f(x)[t] - \mathcal{N}(x)[t]\|_{L^p(K^N)} < \epsilon, \tag{3.40}$$

where $\gamma(\mathrm{ReLU}) = 0$, $\gamma(\sigma) = 1$ *for other non-degenerating activation σ.*

Before we begin the proof of the theorem, we summarize the scheme used in the proof. In [33], an MLP of width $\max\{d_x + 1, d_y\} + \gamma(\sigma)$ approximating a given target function f is constructed using the "encoding scheme." More concretely, the MLP is separated into three parts: encoder, memorizer, and decoder.

First, the encoder part quantizes each component of the input and output into a finite set. The authors use the quantization function $q_n : [0, 1] \to \mathcal{C}_n$

$$q_n(v) := \max\left\{c \in \mathcal{C}_n \;\middle|\; c \leq v\right\}, \tag{3.41}$$

where $\mathcal{C}_n := \{0, 2^{-n}, 2 \times 2^{-n}, \ldots, 1 - 2^{-n}\}$. Then, each quantized vector is encoded into a real number by concatenating its components through the encoder $\mathrm{Enc}_M : [0,1]^{d_x} \to \mathcal{C}_{d_x M}$

$$\mathrm{Enc}_M(x) := \sum_{i=1}^{d_x} q_M(x_i) 2^{-(i-1)M}. \tag{3.42}$$

For small $\delta_1 > 0$, the authors construct an MLP $\mathcal{N}_{enc} : [0,1]^{d_x} \to \mathcal{C}_{d_x M}$ of width $d_x + 1 + \gamma(\sigma)$ satisfying

$$\|\mathrm{Enc}_M(x) - \mathcal{N}_{enc}(x)\| < \delta_1. \tag{3.43}$$

Although the quantization causes a loss in input information, the L^p norm neglects some loss in a sufficiently small domain.

After encoding the input x to $\mathrm{Enc}_M(x)$ with large M, authors use the information of x in $\mathrm{Enc}_M(x)$ to obtain the information of the target output $f(x)$. More precisely, they define the memorizer $\mathrm{Mem}_{M,M'} : \mathcal{C}_{d_x M} \to \mathcal{C}_{d_y M'}$ to map the encoded input $\mathrm{Enc}_M(x)$ to the encoded output $\mathrm{Enc}_{M'}(f(x))$ as

$$\mathrm{Mem}\left(\mathrm{Enc}_M(x)\right) := \left(\mathrm{Enc}_{M'} \circ f \circ q_M\right)(x), \tag{3.44}$$

assuming the quantized map q_M acts on x component-wise in the above equation. Then, an MLP $\mathcal{N}_{mem}$ of width $2 + \gamma(\sigma)$ approximates Mem; that is, for any $\delta_2 > 0$, there exists $\mathcal{N}_{mem}$ satisfying

$$\sup_{x \in [0,1]^{d_x}} \|\mathrm{Mem}(\mathrm{Enc}(x)) - \mathcal{N}_{mem}(\mathrm{Enc}(x))\| < \delta_2. \tag{3.45}$$

Finally, the decoder reconstructs the original output vector from the encoded output vector by cutting off the concatenated components. Owing to the preceding encoder and memorizer, it is enough to define only the value of the decoder on $\mathcal{C}_{d_y M'}$. Hence the decoder $\mathrm{Dec} : \mathcal{C}_{d_y M'} \to \mathcal{C}_{M'}^{d_y} := (\mathcal{C}_{M'})^{d_y}$ is determined by

$$\mathrm{Dec}_{M'}(v) := \hat{v} \qquad \text{where} \qquad \{\hat{v}\} := \mathrm{Enc}_{M'}^{-1}(v) \cap \mathcal{C}_{M'}^{d_y}. \tag{3.46}$$

Indeed in [33], for small $\delta_3 > 0$, an MLP $\mathcal{N}_{dec} : \mathcal{C}_{d_y M'} \to \mathcal{C}_{M'}^{d_y}$ of width $d_y + \gamma(\sigma)$ is construct so that

$$\|\mathrm{Dec}_{M'}(v) - \mathcal{N}_{dec}(v)\| < \delta_3. \tag{3.47}$$

Although (3.43) and (3.47) are not equations but approximations when the activation is just non-degenerate, the composition $\mathcal{N} = \mathcal{N}_{dec} \circ \mathcal{N}_{mem} \circ \mathcal{N}_{enc}$ approximates a target f with sufficiently large M, M' and sufficiently small δ_1, δ_2.

Let us return to the proof of Theorem 3.14. We construct the encoder, memorizer, and decoder similarly. As the encoder and decoder is independent of time t, we use a token-wise MLP and modified RNNs define the token-wise MLPs. On the other hand, the memorizer must work differently according to the time t owing to the multiple output functions. Instead of implementing various memorizers, we separate their input and output domains at each time by translation. Then, it is enough to define one memorizer on the disjoint union of domains.

Proof of Theorem 3.14. We first combine the token-wise encoder and translation for the separation of the domains. Consider the token-wise encoder $\mathrm{Enc}_M : \mathbb{R}^{d_x \times N} \to$

$\mathbb{R}^{1\times N}$, and the following recurrent cells $\mathcal{R}_1, \mathcal{R}_2 : \mathbb{R}^{1\times N} \to \mathbb{R}^{1\times N}$

$$\mathcal{R}_1(v)[t+1] = 2^{-d_x M}\mathcal{R}_1(v)[t] + v[t+1], \tag{3.48}$$

$$\mathcal{R}_2(v)[t+1] = \mathcal{R}_2(v)[t] + 1. \tag{3.49}$$

Then the composition $\mathcal{R}_{enc} = \mathcal{R}_2\mathcal{R}_1 \operatorname{Enc}_M$ defines an encoder of sequence from K^N to $\mathbb{R}^{1\times N}$:

$$\mathcal{R}_{enc}(x)[t] = t + \sum_{j=1}^{t} \operatorname{Enc}_M(x[j])2^{-(j-1)d_x M}, \tag{3.50}$$

where $x = (x[t])_{t=1,\ldots,N}$ is a sequence in K. Note that the range D of $\mathcal{R}_{enc}$ is a disjoint union of compact sets;

$$D = \bigsqcup_{t=1}^{N} \left\{\mathcal{R}_{enc}(x)[t] : x \in K^N\right\}. \tag{3.51}$$

Hence there exists a memorizer $\operatorname{Mem} : \mathbb{R} \to \mathbb{R}$ satisfying

$$\operatorname{Mem}(\mathcal{R}_{enc}(x))[t] = \operatorname{Enc}_{M'}\left(f\left(q_M(x)\right)[t]\right) \tag{3.52}$$

for each $t = 1, 2, \ldots, N$. The token-wise decoder $\operatorname{Dec}_{M'}$ is the last part of the proof.

To complete the proof, we need an approximation of the token-wise encoder $\operatorname{Enc}_M : \mathbb{R}^{d_x} \to \mathbb{R}$, modified recurrent cells $\mathcal{R}_1, \mathcal{R}_2 : \mathbb{R}^{1\times N} \to \mathbb{R}^{1\times N}$, token-wise memorizer $\operatorname{Mem} : \mathbb{R} \to \mathbb{R}$, and token-wise decoder $\operatorname{Dec}_{M'} : \mathbb{R} \to \mathbb{R}^{d_y}$. Following [33], there exist MLPs of width $d_x + 1 + \gamma(\sigma)$, $2 + \gamma(\sigma)$, and $d_y + \gamma(\sigma)$ that approximate Enc_M, Mem, and $\operatorname{Dec}_{M'}$ respectively. Lemma 3.2 shows that $\mathcal{R}_1, \mathcal{R}_2$ is approximated by an RNN of width $2 + \beta(\sigma)$. Hence, an RNN of width $\max\{d_x + 1 + \gamma(\sigma), 2 + \beta(\sigma), 2 + \gamma(\sigma), d_y + \gamma(\sigma)\} = \max\{d_x + 1, d_y\} + \gamma(\sigma)$ approximates the target function f. $\square$

3.4 Variants of RNN

This section describes the universal property of some variants of RNN, particularly LSTM, GRU, or BRNN. LSTM and GRU are proposed to solve the long-term dependency problem. As an RNN has difficulty calculating and updating its parameters for long sequential data, LSTM and GRU take advantage of additional structures in their cells. We prove that they have the same universal property as the original RNN. On the other hand, a BRNN is proposed to overcome the past dependency of an RNN. BRNN consists of two RNN cells, one of which works in reverse order. We prove the universal approximation theorem of a BRNN with the target class of any sequence-to-sequence function.

The universal property of an LSTM originates from the universality of an RNN. Mathematically LSTM $\mathcal{R}_{LSTM}$ indicates a process that computes two outputs, h and c, defined by (3.2). As an LSTM can reproduce an RNN with the same width, we have the following corollary:

Corollary 3.15 (Universal approximation theorem of deep LSTM)**.** *Let $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$ be a continuous past-dependent sequence-to-sequence function. Then, for any $\epsilon > 0$ and compact subset $K \subset \mathbb{R}^{d_x}$, there exists a deep LSTM $\mathcal{N}_{LSTM}$, of width $d_x + d_y + 3$, such that*

$$\sup_{x \in K^N} \sup_{1 \le t \le N} \|f(x)[t] - \mathcal{N}_{LSTM}(x)[t]\| < \epsilon. \tag{3.53}$$

Proof. We set all parameters but W_c, U_c, b_c, and b_f as zeros, and then (3.2) is simplified as

$$
\begin{aligned}
c[t+1] &= \sigma_{\text{sig}}(b_f)c[t] + \frac{1}{2}\tanh\left(U_c h[t] + W_c x[t+1] + b_c\right), \\
h[t+1] &= \frac{1}{2}\tanh\left(c[t+1]\right).
\end{aligned} \tag{3.54}
$$

For any $\epsilon > 0$, a sufficiently large negative b_f yields

$$
\left\| h[t+1] - \frac{1}{2}\tanh\left(\frac{1}{2}\tanh\left(U_c h[t] + W_c x[t+1] + b_c\right)\right) \right\| < \epsilon. \tag{3.55}
$$

Thus, an LSTM reproduces an RNN whose activation function is $\left(\frac{1}{2}\tanh\right) \circ \left(\frac{1}{2}\tanh\right)$ without any additional width in its hidden states. In other words, an LSTM of width d approximates an RNN of width d equipped with the activation function $\left(\frac{1}{2}\tanh\right) \circ \left(\frac{1}{2}\tanh\right)$. $\square$

The universality of GRU is proved similarly.

Corollary 3.16 (Universal approximation theorem of deep GRU)**.** *Let $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$ be a continuous past-dependent sequence-to-sequence function. Then, for any $\epsilon > 0$ and compact subset $K \subset \mathbb{R}^{d_x}$, there exists a deep GRU $\mathcal{N}_{GRU}$, of width $d_x + d_y + 3$, such that*

$$
\sup_{x \in K^N} \sup_{1 \le t \le N} \| f(x)[t] - \mathcal{N}_{GRU}(x)[t] \| < \epsilon. \tag{3.56}
$$

Proof. Setting only W_h, U_h, b_h, and b_z as non-zero, the GRU is simplified as

$$
h[t+1] = \left(1 - \sigma_{\text{sig}}\left(b_z\right)\right) h[t] + \sigma_{\text{sig}}\left(b_z\right) \tanh\left(W_h x[t+1] + \frac{1}{2}U_h h[t] + b_h\right). \tag{3.57}
$$

For any $\epsilon > 0$, a sufficiently large b_z yields

$$\left\| h[t+1] - \tanh\left(W_h x[t+1] + \frac{1}{2} U_h h[t] + b_h \right) \right\| < \epsilon. \tag{3.58}$$

Hence, we attain the corollary. □

Remark 3.17. *We refer to the width as the maximum of hidden states. However, the definition is somewhat inappropriate, as LSTM and GRU cells have multiple hidden states; hence, there are several times more components than an RNN with the same width. Thus we expect that they have better approximation power or have a smaller minimum width for universality than an RNN. Nevertheless, we retain the theoretical proof as future work to identify whether they have different abilities in approximation or examine why they exhibit different performances in practical applications.*

Now, let us focus on the universality of a BRNN. Recall that a stack of modified recurrent cells $\mathcal{N}$ construct a linear combination of the previous input components $x[1:t]$ at each time,

$$\mathcal{N}(x)[t] = \begin{bmatrix} x[t] \\ \sum_{j=1}^{t} A_j[t] x[j] \end{bmatrix}. \tag{3.59}$$

Therefore, if we reverse the order of sequence and flow of the recurrent structure, a stack of reverse modified recurrent cells $\bar{\mathcal{N}}$ constructs a linear combination of the subsequent input components $x[t:N]$ at each time,

$$\bar{\mathcal{N}}(x)[t] = \begin{bmatrix} x[t] \\ \sum_{j=t}^{N} B_j[t] x[j] \end{bmatrix}. \tag{3.60}$$

From this point of view, we expect that a stacked BRNN successfully approximates an arbitrary sequence-to-sequence function beyond the past dependency. As previously mentioned, we prove it in the following lemma.

Lemma 3.18. *Suppose $A_j[t] \in \mathbb{R}^{1\times d_x}$ are the given matrices for $1 \le t \le N$, $1 \le j \le N$. Then there exists a modified TBRNN $\tilde{\mathcal{N}} : \mathbb{R}^{d_x\times N} \to \mathbb{R}^{(d_x+1)\times N}$ of width d_x+1 such that*

$$\tilde{\mathcal{N}}(x)[t] = \begin{bmatrix} x[t] \\ \sigma\left(\sum_{j=1}^{N} A_j[t]x[j]\right) \end{bmatrix}, \tag{3.61}$$

for all $t = 1, 2, \ldots, N$.

Sketch of proof. The detailed proof is available in Section 3.6.5. We use modified TBRNN cells with either only a forward modified TRNN or a backward modified TRNN. The stacked forward modified TRNN cells compute $\sum_{j=1}^{t} A_j[t]x[j]$, and the stacked backward modified TRNN cells compute $\sum_{j=t+1}^{N} A_j[t]x[j]$. □

As in previous cases, we have the following theorem for a TBRNN. The proof is almost the same as that of Lemma 3.11 and 3.6.

Lemma 3.19. *Suppose $w_i \in \mathbb{R}^R$, $A_{i,j}[t] \in \mathbb{R}^{1\times d_x}$ are the given matrices for $1 \le t \le N$, $1 \le j \le N$, $1 \le i \le M$. Then there exists a modified TBRNN $\tilde{\mathcal{N}} : \mathbb{R}^{d_x\times N} \to \mathbb{R}^{1\times N}$ of width d_x+2 such that*

$$\tilde{\mathcal{N}}(x)[t] = \sum_{i=1}^{M} w_i\sigma\left(\sum_{j=1}^{N} A_{i,j}[t]x[j]\right). \tag{3.62}$$

Proof. First, construct a modified deep TBRNN $\mathcal{N}_1 : \mathbb{R}^{d_x\times N} \to \mathbb{R}^{(d_x+2)\times N}$ of width

$d_x + 2$ such that

$$\mathcal{N}_1(x)[t] = \begin{bmatrix} x[t] \\ \sigma\left(\sum_{j=1}^{N} A_{1,j}[t]x[j]\right) \\ 0 \end{bmatrix}, \tag{3.63}$$

as Lemma 3.18. The final component does not affect the first linear summation and remains zero. After $\mathcal{N}_1$, use the (d_x+2)th component to obtain a stack of cells $\mathcal{N}_2 : \mathbb{R}^{(d_x+2)\times N} \to \mathbb{R}^{(d_x+2)\times N}$, which satisfies

$$\mathcal{N}_2\mathcal{N}_1(x)[t] = \begin{bmatrix} x[t] \\ \sigma\left(\sum_{j=1}^{N} A_{1,j}[t]x[j]\right) \\ \sigma\left(\sum_{j=1}^{N} A_{2,j}[t]x[j]\right) \end{bmatrix}, \tag{3.64}$$

and use a modified RNN cell $\mathcal{R}$ to sum up the results and reset the last component:

$$\mathcal{R}\mathcal{N}_2\mathcal{N}_1(x)[t] = \begin{bmatrix} x[t] \\ w_1\sigma\left(\sum_{j=1}^{N} A_{1,j}[t]x[j]\right) + w_2\sigma\left(\sum_{j=1}^{N} A_{2,j}[t]x[j]\right) \\ 0 \end{bmatrix}. \tag{3.65}$$

As the (d_x+2)th component resets to zero, we use it to compute the third sum $w_3\sigma\left(\sum A_{3,j}[t]x[j]\right)$ and repeat until we obtain the final network $\mathcal{N}$ such that

$$\mathcal{N}(x)[t] = \begin{bmatrix} x[t] \\ \sum_{i=1}^{M} w_i\sigma\left(\sum_{t=1}^{N} A_{i,j}[t]x[j]\right) \\ 0 \end{bmatrix}. \tag{3.66}$$

$\square$

The following lemma fills the gap between a modified TBRNN and a modified

BRNN.

Lemma 3.20. *Let $\tilde{\mathcal{N}}$ be a modified TBRNN that computes* (3.61) *and $K \subset \mathbb{R}^{d_x}$ be a compact set. Then for any $\epsilon > 0$ there exists a modified BRNN $\bar{\mathcal{N}}$ of width $d_x + 2 + \gamma(\sigma)$ such that*

$$\sup_{x \in K^N} \left\| \tilde{\mathcal{N}}(x) - \bar{\mathcal{N}}(x) \right\| < \epsilon, \tag{3.67}$$

where $\gamma(\mathrm{ReLU}) = 0$, $\gamma(\sigma) = 1$ for non-degenerating activation σ.
Moreover, there exists a BRNN $\mathcal{N}$ of width $d_x + 2 + \alpha(\sigma)$ such that

$$\sup_{x \in K^N} \left\| \tilde{\mathcal{N}}(x) - \mathcal{N}(x) \right\| < \epsilon, \tag{3.68}$$

where

$$\alpha(\sigma) = \begin{cases} 0 & \sigma \textit{ is ReLU}, \\ 1 & \sigma \textit{ is non-degenerating function with } \sigma(z_0) = 0, \\ 2 & \sigma \textit{ is non-degenerating function with } \sigma(z_0) \neq 0. \end{cases} \tag{3.69}$$

Proof. We omit these details because we only need to construct a modified RNN that approximates (3.59) and (3.60) using Lemma 3.12. As only the forward or backward modified RNN cell is used in the proof of Lemma 3.18, it is enough for the modified BRNN to approximate either the forward or backward modified TRNN. Thus, it follows from Lemma 3.12. Lemma 3.2 provides the second part of this theorem. □

Finally, we obtain the universal approximation theorem of the BRNN from the previous results.

Theorem 3.21 (Universal approximation theorem of deep BRNN)**.** *Let $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{d_y \times N}$ be a continuous sequence to seqeunce function and σ be a non-degenerate activation function. Then for any $\epsilon > 0$ and compact subset $K \subset \mathbb{R}^{d_x}$, there exists a deep BRNN $\mathcal{N}$ of width $d_x + d_y + 2 + \alpha(\sigma)$, such that*

$$\sup_{x \in K^N} \sup_{1 \le t \le N} \|f(x)[t] - \mathcal{N}(x)[t]\| < \epsilon, \tag{3.70}$$

where

$$\alpha(\sigma) = \begin{cases} 0 & \sigma \text{ is ReLU,} \\ 1 & \sigma \text{ is non-degenerating function with } \sigma(z_0) = 0, \\ 2 & \sigma \text{ is non-degenerating function with } \sigma(z_0) \neq 0. \end{cases} \tag{3.71}$$

Proof. As in the proof of Theorem 3.4, we set $d_y = 1$ for notational convenience. According Lemma 3.19, there exists a modified TBRNN $\tilde{\mathcal{N}}$ of width $d_x + 2$ such that

$$\sup_{x \in K^n} \left\| f(x) - \tilde{\mathcal{N}}(x) \right\| < \frac{\epsilon}{2}. \tag{3.72}$$

Lemma 3.20 implies that there exists a BRNN of width $d_x + 3 + \alpha(\sigma)$ such that

$$\sup_{x \in K^n} \left\| \tilde{\mathcal{N}}(x) - \mathcal{N}(x) \right\| < \frac{\epsilon}{2}. \tag{3.73}$$

The triangle inequality leads to

$$\sup_{x \in K^n} \|f(x) - \mathcal{N}(x)\| < \epsilon. \tag{3.74}$$

□

3.5 Discussion

We proved the universal approximation theorem and calculated the upper bound of the minimum width of an RNN, an LSTM, a GRU, and a BRNN. In this section, we illustrate how our results support the performance of a recurrent network.

We show that an RNN needs a width of at most $d_x + d_y + 4$ to approximate a function from a sequence of d_x-dimensional vectors to a sequence of d_y-dimensional vectors. The upper bound of the minimum width of the network depends only on the input and output dimensions, regardless of the length of the sequence. The independence of the sequence length indicates that the recurrent structure is much more effective in learning a function on sequential inputs. To approximate a function defined on a long sequence, a network with a feed-forward structure requires a wide width proportional to the length of the sequence. For example, an MLP should have a wider width than Nd_x if it approximates a function $f : \mathbb{R}^{d_x \times N} \to \mathbb{R}$ defined on a sequence [20]. However, with the recurrent structure, it is possible to approximate via a narrow network of width d_x+1 regardless of the length, because the minimum width is independent of the length N. This suggests that the recurrent structure, which transfers information between different time steps in the same layer, is crucial for success with sequential data.

From a practical point of view, this fact further implies that there is no need to limit the length of the time steps that affect dynamics to learn the internal dynamics between sequential data. For instance, suppose that a pair of long sequential data $(x[t])$ and $(y[t])$ have an unknown relation $y[t] = f\left(x[t-p], x[t-p+1], \ldots, x[t]\right)$. Even without prior knowledge of f and p, a deep RNN learns the relation if we train the network with inputs $x[1:t]$ and outputs $y[t]$. The MLP cannot reproduce the result because the required width increases proportionally to p, which is an unknown factor. The difference between these networks theoretically supports that recurrent networks are appropriate when dealing with sequential data whose underlying dynamics are unknown in the real world.

3.6 Proofs

3.6.1 Proof of the Lemma 3.2

Without loss of generality, we may assume $\bar{\mathcal{P}}$ is an identity map and $I = \{1, 2, \dots, k\}$. Let $\bar{\mathcal{R}}(x)[t+1] = \sigma_I\left(\bar{A}\mathcal{R}(x)[t] + \bar{B}x[t+1] + \bar{\theta}\right)$ be a given modified RNN cell, and $\mathcal{Q}(x)[t] = \bar{Q}x[t]$ be a given token-wise linear projection map. We use notations $O_{m,n}$ and $\mathbf{1}_m$ to denote zero matrix in $\mathbb{R}^{m\times n}$ and one vector in $\mathbb{R}^m$ respectively. Sometimes we omit $O_{m,n}$ symbol in some block-diagonal matrices if the size of the zero matrix is clear.

Case 1: $\sigma(z_0) = 0$

Let $\mathcal{P}$ be the identity map. For $\delta > 0$ define $\mathcal{R}_1^\delta$ as

$$\mathcal{R}_1^\delta\left(x[t+1]\right) \coloneqq \sigma\left(\delta\bar{B}x[t+1] + \delta\bar{\theta} + z_0\mathbf{1}_d\right). \tag{3.75}$$

Since σ is non-degenerating at z_0 and σ' is continuous at z_0, we have

$$\mathcal{R}_1^\delta \circ \mathcal{P}(x)[t+1] = \delta\sigma'(z_0)\left(\bar{B}x[t+1] + \bar{\theta}\right) + o(\delta). \tag{3.76}$$

Then construct a second cell to compute transition as

$$\mathcal{R}_2^\delta(x)[t+1]$$
$$= \sigma\left(\tilde{A}\mathcal{R}_2^\delta(x)[t] + \frac{1}{\sigma'(z_0)}\begin{bmatrix}\delta^{-1}I_k & \\ & I_{d-k}\end{bmatrix} x[t+1] + \begin{bmatrix}\mathbf{0}_k \\ z_0\mathbf{1}_{d-k}\end{bmatrix}\right), \quad (3.77)$$

where $\tilde{A} = \begin{bmatrix}I_k & \\ & \delta I_{d-k}\end{bmatrix}\bar{A}\begin{bmatrix}I_k & \\ & \frac{1}{\delta\sigma'(z_0)}I_{d-k}\end{bmatrix}$.

After that, the first output of $\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)$ becomes

$$\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)[1] = \sigma\left(\frac{1}{\sigma'(z_0)}\begin{bmatrix}\delta^{-1}I_k & \\ & I_{d-k}\end{bmatrix}\mathcal{R}_1^\delta(x)[1] + \begin{bmatrix}\mathbf{0}_k \\ z_0\mathbf{1}_{d-k}\end{bmatrix}\right) \quad (3.78)$$
$$= \sigma\left(\begin{bmatrix}\left(\bar{B}x[1]+\bar{\theta}\right)_{1:k} + \delta^{-1}o(\delta) \\ \left(z_0\mathbf{1}_{d-k} + \delta(\bar{B}x[1]+\bar{\theta}\right)_{k+1:d} + o(\delta)\end{bmatrix}\right) \quad (3.79)$$
$$= \begin{bmatrix}\sigma\left(\bar{B}x[1]+\bar{\theta}\right)_{1:k} + o(1) \\ \sigma'(z_0)\delta\left(\bar{B}x[1]+\bar{\theta}\right)_{k+1:d} + o(\delta)\end{bmatrix} \quad (3.80)$$
$$= \begin{bmatrix}\bar{\mathcal{R}}(x)[1]_{1:k} + o(1) \\ \sigma'(z_0)\delta\bar{\mathcal{R}}(x)[1]_{k+1:d} + o(\delta)\end{bmatrix}. \quad (3.81)$$

Now use mathematical induction on time t to compute $\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)$ assuming

$$\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)[t] = \begin{bmatrix}\bar{\mathcal{R}}(x)[t]_{1:k} + o(1) \\ \sigma'(z_0)\delta\bar{\mathcal{R}}(x)[t]_{k+1:d} + o(\delta)\end{bmatrix}. \quad (3.82)$$

From a direct calculation, we attain

$$\frac{1}{\sigma'(z_0)}\begin{bmatrix}\delta^{-1}I_k & \\ & I_{d-k}\end{bmatrix}\mathcal{R}_1^{\delta}\mathcal{P}(x)[t+1]+\begin{bmatrix}\mathbf{0}_k\\ z_0\mathbf{1}_{d-k}\end{bmatrix} \tag{3.83}$$

$$=\frac{1}{\sigma'(z_0)}\begin{bmatrix}\delta^{-1}I_k & \\ & I_{d-k}\end{bmatrix}\left(\delta\sigma'(z_0)\left(\bar{B}x[t+1]+\bar{\theta}\right)+o(\delta)\right)+\begin{bmatrix}\mathbf{0}_k\\ z_0\mathbf{1}_{d-k}\end{bmatrix} \tag{3.84}$$

$$=\begin{bmatrix}\bar{B}x[t+1]_{1:k}+\bar{\theta}_{1:k}+\delta^{-1}o(\delta)\\ z_0\mathbf{1}_{d-k}+\delta\left(\bar{B}x[t+1]+\bar{\theta}\right)_{k+1:d}+o(\delta)\end{bmatrix}, \tag{3.85}$$

and

$$\tilde{A}\mathcal{R}_2^{\delta}\mathcal{R}_1^{\delta}\mathcal{P}(x)[t] \tag{3.86}$$

$$=\begin{bmatrix}I_k & \\ & \delta I_{d-k}\end{bmatrix}\bar{A}\begin{bmatrix}I_k & \\ & \frac{1}{\delta\sigma'(z_0)}I_{d-k}\end{bmatrix}\begin{bmatrix}\bar{\mathcal{R}}(x)[t]_{1:k}+o(1)\\ \sigma'(z_0)\delta\bar{\mathcal{R}}(x)[t]_{k+1:d}+o(\delta)\end{bmatrix} \tag{3.87}$$

$$=\begin{bmatrix}I_k & \\ & \delta I_{d-k}\end{bmatrix}\bar{A}\begin{bmatrix}\bar{\mathcal{R}}(x)[t]_{1:k}+o(1)\\ \bar{\mathcal{R}}(x)[t]_{k+1:d}+o(1)\end{bmatrix} \tag{3.88}$$

$$=\begin{bmatrix}\left(\bar{A}\bar{\mathcal{R}}(x)[t]\right)_{1:k}+o(1)\\ \delta\left(\bar{A}\bar{\mathcal{R}}(x)[t]\right)_{k+1:d}+o(\delta)\end{bmatrix}. \tag{3.89}$$

With the sum of above two results, we obtain the induction hypothesis (3.82) for

$t+1$,

$$\mathcal{R}_2^\delta \mathcal{R}_1^\delta \mathcal{P}(x)[t+1] \tag{3.90}$$

$$= \sigma \left(\tilde{A} \mathcal{R}_2^\delta \mathcal{R}_1^\delta \mathcal{P}(x)[t] + \frac{1}{\sigma'(z_0)} \begin{bmatrix} \frac{I_k}{\delta} & \\ & I_{d-k} \end{bmatrix} \mathcal{R}_1^\delta \mathcal{P}(x)[t+1] + \begin{bmatrix} \mathbf{0}_k \\ z_0 \mathbf{1}_{d-k} \end{bmatrix} \right) \tag{3.91}$$

$$= \sigma \begin{bmatrix} \left(\bar{A}\bar{\mathcal{R}}(x)[t]\right)_{1:k} + \bar{B}x[t+1]_{1:k} + \bar{\theta}_{1:k} + o(1) \\ z_0 \mathbf{1}_{d-k} + \delta \left(\bar{A}\bar{\mathcal{R}}(x)[t]\right)_{k+1:d} + \delta \left(\bar{B}x[t+1] + \bar{\theta}\right)_{k+1:d} + o(\delta) \end{bmatrix} \tag{3.92}$$

$$= \begin{bmatrix} \bar{\mathcal{R}}(x)[t+1]_{1:k} + o(1) \\ \sigma'(z_0)\delta \bar{\mathcal{R}}(x)[t+1]_{k+1:d} + o(\delta) \end{bmatrix}. \tag{3.93}$$

Setting $\mathcal{Q}^\delta = \bar{Q} \begin{bmatrix} I_k & \\ & \frac{1}{\sigma'(z_0)\delta} I_{d-k} \end{bmatrix}$ and choosing δ small enough complete the proof:

$$\mathcal{Q}^\delta \mathcal{R}_2^\delta \mathcal{R}_1^\delta \mathcal{P}(x)[t] = \bar{Q} \begin{bmatrix} \bar{\mathcal{R}}(x)[t]_{1:k} + o(1) \\ \bar{\mathcal{R}}(x)[t]_{k+1:d} + o(1) \end{bmatrix} = \bar{\mathcal{Q}}\bar{\mathcal{R}}(x)[t] + o(1) \to \bar{\mathcal{Q}}\bar{\mathcal{R}}(x)[t]. \tag{3.94}$$

Case 2: $\sigma(z_0) \neq 0$

When $\sigma(z_0) \neq 0$, there is $\sigma(z_0)$ term independent of δ in the Taylor expansion of $\sigma(z_0 + \delta x) = \sigma(z_0) + \delta\sigma'(z_0)x + o(\delta)$. An additional width removes the term in this case; hence we need a lifting map $\mathcal{P} : \mathbb{R}^{d\times N} \to \mathbb{R}^{(d+1)\times N}$:

$$\mathcal{P}(x)[t] = \begin{bmatrix} x[t] \\ 0 \end{bmatrix}. \tag{3.95}$$

Now for $\delta > 0$ define $\mathcal{R}_1^\delta$ as

$$\mathcal{R}_1^\delta(X) \coloneqq \sigma\left(\delta\begin{bmatrix}\bar{B} & \\ & 0\end{bmatrix}X + \delta\begin{bmatrix}\bar{\theta}\\ 0\end{bmatrix} + z_0\mathbf{1}_{d+1}\right). \tag{3.96}$$

As in the previous case, we have

$$\mathcal{R}_1^\delta \circ \mathcal{P}(x)[t+1] = \begin{bmatrix}\sigma(z_0)\mathbf{1}_d + \delta\sigma'(z_0)\left(\bar{B}x[t+1] + \bar{\theta}\right) + o(\delta)\\ \sigma(z_0)\end{bmatrix}, \tag{3.97}$$

and construct a second cell $\mathcal{R}_2^\delta$ to compute

$$\begin{aligned}\mathcal{R}_2^\delta(x)[t+1] = \sigma\Bigg(\tilde{A}\mathcal{R}_2^\delta(x)[t] + \begin{bmatrix}\frac{I_k}{\delta\sigma'(z_0)} & \\ & \frac{I_{d+1-k}}{\sigma'(z_0)}\end{bmatrix}x[t+1]\\ + \begin{bmatrix}-\frac{\sigma(z_0)}{\delta\sigma'(z_0)}\mathbf{1}_k\\ z_0\mathbf{1}_{d+1-k} - \frac{\sigma(z_0)}{\sigma'(z_0)}\mathbf{1}_{d+1-k}\end{bmatrix}\Bigg),\end{aligned} \tag{3.98}$$

where $\tilde{A} = \begin{bmatrix}I_k & \\ & \delta I_{d+1-k}\end{bmatrix}\begin{bmatrix}\bar{A} & \\ & 0\end{bmatrix}\begin{bmatrix}I_k & & \\ & \frac{1}{\delta\sigma'(z_0)}I_{d-k} & -\frac{1}{\delta\sigma'(z_0)}\mathbf{1}_{d-k}\\ & & 0\end{bmatrix}$.

After that, the first output of $\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)[t]$ becomes

$$\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)[1] = \sigma\left(\frac{1}{\sigma'(z_0)}\begin{bmatrix}\delta^{-1}I_k & \\ & I_{d+1-k}\end{bmatrix}\mathcal{R}_1^\delta(x)[1]\right. \tag{3.99}$$

$$\left. + \begin{bmatrix} -\frac{\sigma(z_0)}{\delta\sigma'(z_0)}\mathbf{1}_k \\ z_0\mathbf{1}_{d+1-k} - \frac{\sigma(z_0)}{\sigma'(z_0)}\mathbf{1}_{d+1-k}\end{bmatrix}\right) \tag{3.100}$$

$$= \sigma\left(\begin{bmatrix}\left(\bar{B}x[1]+\bar{\theta}\right)_{1:k} + o(\delta) \\ \left(z_0\mathbf{1}_{d-k} + \delta(\bar{B}x[1]+\bar{\theta}\right)_{k+1:d} + o(\delta) \\ z_0\end{bmatrix}\right) \tag{3.101}$$

$$= \begin{bmatrix}\sigma\left(\bar{B}x[1]+\bar{\theta}\right)_{1:k} + o(1) \\ \sigma(z_0)\mathbf{1}_{d-k} + \sigma'(z_0)\delta\left(\bar{B}x[1]+\bar{\theta}\right)_{k+1:d} + o(\delta) \\ \sigma(z_0)\end{bmatrix} \tag{3.102}$$

$$= \begin{bmatrix}\bar{\mathcal{R}}(x)[1]_{1:k} + o(1) \\ \sigma(z_0)\mathbf{1}_{d-k} + \sigma'(z_0)\delta\bar{\mathcal{R}}(x)[1]_{k+1:d} + o(\delta) \\ \sigma(z_0)\end{bmatrix}. \tag{3.103}$$

Assume $\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)$ and use mathematical induction on time t.

$$\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)[t] = \begin{bmatrix}\bar{\mathcal{R}}(x)[t]_{1:k} + o(1) \\ \sigma(z_0)\mathbf{1}_{d-k} + \sigma'(z_0)\delta\bar{\mathcal{R}}(x)[t]_{k+1:d} + o(\delta) \\ \sigma(z_0)\end{bmatrix}. \tag{3.104}$$

Direct calculation yields

$$\frac{1}{\sigma'(z_0)}\begin{bmatrix}\delta^{-1}I_k & \\ & I_{d+1-k}\end{bmatrix}\mathcal{R}_1^\delta\mathcal{P}(x)[t+1]+\begin{bmatrix}-\frac{\sigma(z_0)}{\delta\sigma'(z_0)}\mathbf{1}_k \\ z_0\mathbf{1}_{d+1-k}-\frac{\sigma(z_0)}{\sigma'(z_0)}\mathbf{1}_{d+1-k}\end{bmatrix} \tag{3.105}$$

$$=\begin{bmatrix}\bar{B}x[t+1]_{1:k}+\bar{\theta}_{1:k}+o(1) \\ z_0\mathbf{1}_{d-k}+\delta\left(\bar{B}x[t+1]+\bar{\theta}\right)_{k+1:d}+o(\delta) \\ z_0\end{bmatrix}, \tag{3.106}$$

and

$$\tilde{A}\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)[t] \tag{3.107}$$

$$=\tilde{A}\begin{bmatrix}\bar{\mathcal{R}}(x)[t]_{1:k}+o(1) \\ \sigma(z_0)\mathbf{1}_{d-k}+\sigma'(z_0)\delta\bar{\mathcal{R}}(x)[t]_{k+1:d}+o(\delta) \\ \sigma(z_0)\end{bmatrix} \tag{3.108}$$

$$=\begin{bmatrix}I_k & \\ & \delta I_{d+1-k}\end{bmatrix}\begin{bmatrix}\bar{A} & \\ & 0\end{bmatrix}\begin{bmatrix}\bar{\mathcal{R}}(x)[t]_{1:k}+o(1) \\ \bar{\mathcal{R}}(x)[t]_{k+1:d}+o(1) \\ 0\end{bmatrix} \tag{3.109}$$

$$=\begin{bmatrix}\left(\bar{A}\bar{\mathcal{R}}(x)[t]\right)_{1:k}+o(1) \\ \delta\left(\bar{A}\bar{\mathcal{R}}(x)[t]\right)_{k+1:d}+o(\delta) \\ 0\end{bmatrix}. \tag{3.110}$$

Adding two terms in (3.98), we obtain the induction hypothesis (3.104) for $t+1$,

$$\mathcal{R}_2^\delta\mathcal{R}_1^\delta\mathcal{P}(x)[t+1]=\begin{bmatrix}\bar{\mathcal{R}}(x)[t+1]_{1:k}+o(1) \\ \sigma(z_0)\mathbf{1}_{d-k}+\sigma'(z_0)\delta\bar{\mathcal{R}}(x)[t+1]_{k+1:d}+o(\delta) \\ \sigma(z_0)\end{bmatrix}. \tag{3.111}$$

Setting $\mathcal{Q}^\delta = \begin{bmatrix} \bar{Q} & 0 \end{bmatrix} \begin{bmatrix} I_k & & \\ & \frac{1}{\sigma'(z_0)\delta} I_{d-k} & -\frac{1}{\sigma'(z_0)\delta} \mathbf{1}_{d-k} \\ & & 0 \end{bmatrix}$ and choosing δ small enough complete the proof:

$$\mathcal{Q}^\delta \mathcal{R}_2^\delta \mathcal{R}_1^\delta \mathcal{P}(x)[t] = \begin{bmatrix} \bar{Q} & 0 \end{bmatrix} \begin{bmatrix} \bar{\mathcal{R}}(x)[t]_{1:k} + o(1) \\ \bar{\mathcal{R}}(x)[t]_{k+1:d} + o(1) \\ 0 \end{bmatrix} = \bar{\mathcal{Q}} \bar{\mathcal{R}}(x)[t] + o(1)$$

$$\to \bar{\mathcal{Q}} \bar{\mathcal{R}}(x)[t]. \quad (3.112)$$

3.6.2 Proof of the Lemma 3.5

It suffices to show that there exists a modified RNN $\mathcal{N}$ that computes

$$\mathcal{N}(x)[N] = \begin{bmatrix} x[N] \\ \sum_{t=1}^N A[t]x[t] \end{bmatrix}, \quad (3.113)$$

for given matrices $A[1], \ldots, A[N] \in \mathbb{R}^{1 \times d_x}$.

RNN should have multiple layers to implement the arbitrary linear combination. To overcome the complex time dependency deriving from deep structures and explicitly formulate the results of deep modified RNN, we force A and B to use the information of the previous time step in a limited way. Define the modified RNN cell at l-th layer $\mathcal{R}_l$ as

$$\mathcal{R}_l(x)[t+1] = A_l \mathcal{R}_l(x)[t] + B_l x[t+1], \quad (3.114)$$

where $A_l = \begin{bmatrix} O_{d_x,d_x} & O_{d_x,1} \\ O_{1,d_x} & 1 \end{bmatrix}$, $B_l = \begin{bmatrix} I_{d_x} & O_{d_x,1} \\ b_l & 1 \end{bmatrix}$ for $b_l \in \mathbb{R}^{1\times d_x}$.

Construct a modified RNN $\mathcal{N}_L$ for each $L \in \mathbb{N}$ as

$$\mathcal{N}_L := \mathcal{R}_L \circ \mathcal{R}_{L-1} \circ \cdots \circ \mathcal{R}_1, \tag{3.115}$$

and denote the output of $\mathcal{N}_L$ at each time m for an input sequence $x' = \begin{bmatrix} x \\ 0 \end{bmatrix} \in \mathbb{R}^{d_x+1}$ of embedding of x:

$$T(n,m) := \mathcal{N}_n\left(x'\right)[m]. \tag{3.116}$$

Then we have the following lemma.

Lemma 3.22. *Let $T(n,m)$ be the matrix defined by* (3.116)*. Then we have*

$$T(n,m) = \begin{bmatrix} x[m] \\ \sum_{i=1}^{\infty}\sum_{j=1}^{\infty}\binom{n+m-i-j}{n-i} b_i x[j] \end{bmatrix}, \tag{3.117}$$

where $\binom{n}{k}$ means binomial coefficient $\frac{n!}{k!(n-k)!}$ for $n \geq k$. We define $\binom{n}{k} = 0$ for the case of $k > n$ or $n < 0$ for notational convenience.

Proof. Since there is no activation in modified RNN (3.114), $T(n,m)$ has the form of

$$T(n,m) = \begin{bmatrix} x_m \\ \sum_{i=1}^{\infty}\sum_{j=1}^{\infty}\alpha_{i,j}^{n,m} b_i x[j] \end{bmatrix}. \tag{3.118}$$

From the definition of the modified RNN cell and T, we first show that α satisfies the recurrence relation

$$\alpha_{i,j}^{n,m} = \begin{cases} \alpha_{i,j}^{n-1,m} + \alpha_{i,j}^{n,m-1} + 1 & \text{if } n = i \text{ and } m = j, \\ \alpha_{i,j}^{n-1,m} + \alpha_{i,j}^{n,m-1} & \text{otherwise.} \end{cases} \tag{3.119}$$

using mathematical induction on n, m in turn. Initially, $T(0, m) = \begin{bmatrix} x_m \\ 0 \end{bmatrix}$, $T(n, 0) = \begin{bmatrix} O_{d_x,1} \\ 0 \end{bmatrix}$ by definition, and (3.118) holds when $n = 0$. Now assume (3.118) holds for $n \leq N$, any m. To show that (3.118) holds for $n = N + 1$ and any m, use mathematical induction on m. By definition, we have $\alpha_{i,j}^{n,0} = 0$ for any n. Thus (3.118) holds when $n = N + 1$ and $m = 0$. Assume it holds for $n = N + 1$ and $m \leq M$. Then

$$\begin{aligned} & T(N+1, M+1) \\ &= \begin{bmatrix} O_{d_x,d_x} & O_{d_x,1} \\ O_{1,d_x} & 1 \end{bmatrix} \begin{bmatrix} x_M \\ \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \alpha_{i,j}^{N+1,M} x_j \end{bmatrix} \\ &\qquad + \begin{bmatrix} I_{d_x} & O_{d_x,1} \\ b_{N+1} & 1 \end{bmatrix} \begin{bmatrix} x_{M+1} \\ \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \alpha_{i,j}^{N,M+1} x_j \end{bmatrix} \\ &= \begin{bmatrix} O_{d_x,1} \\ \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \alpha_{i,j}^{N+1,M} b_i x_j \end{bmatrix} \\ &\qquad + \begin{bmatrix} x_{M+1} \\ b_{N+1} x_{M+1} + \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \left(\alpha_{i,j}^{N+1,M} + \alpha_{i,j}^{N,M+1} \right) b_i x_j \end{bmatrix}. \end{aligned} \tag{3.120}$$

Hence the relation holds for $n = N + 1$ and any $m > 0$.

Now remains to show

$$\alpha_{i,j}^{n,m} = \begin{cases} \binom{m+n-i-j}{n-i} & \text{if } 1 \le i \le n,\ 1 \le j \le m, \\ 0 & \text{otherwise.} \end{cases} \tag{3.121}$$

From the initial condition of α, we know $\alpha_{i,j}^{0,m} = \alpha_{i,j}^{n,0} = 0$ for all $n, m \in \mathbb{N}$. After some direct calculation with the recurrence relation (3.118) of α, we have

i) If $n < i$ or $m < j$, $\alpha_{i,j}^{n,m} = 0$ as $\alpha_{i,j}^{n,m} = \alpha_{i,j}^{n-1,m} + \alpha_{i,j}^{n,m-1}$.

ii) $\alpha_{i,j}^{i,j} = \alpha_{i,j}^{i-1,j} + \alpha_{i,j}^{i,j-1} + 1 = 1$.

iii) $\alpha_{i,j}^{i,m} = \alpha_{i,j}^{i-1,m} + \alpha_{i,j}^{i,m-1} = \alpha_{i,j}^{i,m-1}$ implies $\alpha_{i,j}^{i,m} = 1$ for $m > j$.

iv) Similarly, $\alpha_{i,j}^{n,j} = \alpha_{i,j}^{n-1,j} + \alpha_{i,j}^{n,j-1} = \alpha_{i,j}^{n-1,j}$ implies $\alpha_{i,j}^{n,j} = 1$ for $n > i$.

Now use mathematical induction on $n+m$ starting from $n+m = i+j$ to show $\alpha_{i,j}^{n,m} = \binom{m+n-i-j}{n-i}$ for $n \ge i$, $m \ge j$.

i) $n + m = i + j$ holds only if $n = i$, $m = j$ for $n \ge i$, $m \ge j$. In the case, $\alpha_{i,j}^{i,j} = 1 = \binom{m+n-i-j}{n-i}$.

ii) Assume that (3.121) holds for any n, m with $n+m = k$ as induction hypothesis. Now suppose $n+m = k+1$ for given n, m. If $n = i$ or $m = j$ we already know $\alpha_{i,j}^{n,m} = 1 = \binom{m+n-i-j}{n-i}$. Otherwise $n-1 \ge i$, $m-1 \ge j$, and we have

$$\begin{aligned} \alpha_{i,j}^{n,m} &= \alpha_{i,j}^{n-1,m} + \alpha_{i,j}^{n,m-1} \\ &= \binom{m+n-1-i-j}{n-1-i} + \binom{m+n-1-i-j}{n-i} \\ &= \binom{m+n-i-j}{n-i}, \end{aligned} \tag{3.122}$$

which completes the proof.

$\square$

We have computed the output of modified RNN $\mathcal{N}_N$ such that

$$\mathcal{N}_N\left(x'\right)[N] = \begin{bmatrix} x[N] \\ \sum_{i=1}^{N}\sum_{j=1}^{N}\binom{2n-i-j}{n-i} b_i x[j] \end{bmatrix}. \tag{3.123}$$

If the square matrix $\Lambda_N = \left\{\binom{2n-i-j}{n-i}\right\}_{1\le i,j\le N}$ has inverse $\Lambda_N^{-1} = \{\lambda_{i,j}\}_{1\le i,j\le N}$, $b_i = \sum_{t=1}^{N}\lambda_{t,i}A[t]$ satisfies

$$\begin{aligned}\sum_{i=1}^{N}\sum_{j=1}^{N}\binom{2n-i-j}{n-i} b_i x[j] &= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{t=1}^{n}\binom{2n-i-j}{n-i}\lambda_{t,i}A[t]x[j] \\ &= \sum_{j=1}^{n}\sum_{t=1}^{n}\left[\sum_{i=1}^{n}\binom{2n-i-j}{n-i}\lambda_{t,i}\right]A[t]x[j] \\ &= \sum_{j=1}^{n}\sum_{t=1}^{n}\delta_{j,t}A[t]x[j] \\ &= \sum_{j=1}^{n}A[j]x[j],\end{aligned} \tag{3.124}$$

where δ is the Kronecker delta function.

The following lemma completes the proof.

Lemma 3.23. *Matrix* $\Lambda_n = \left\{\binom{2n-i-j}{n-i}\right\}_{1\le i,j\le n} \in \mathbb{R}^{n\times n}$ *is invertible.*

Proof. Use mathematical induction on n. Λ_1 is a trivial case. Assume Λ_n is invertible.

$$\Lambda_{n+1} = \begin{bmatrix} \binom{2n}{n} & \binom{2n-1}{n} & \binom{2n-2}{n} & \dots & \binom{n+1}{n} & \binom{n}{n} \\ \binom{2n-1}{n-1} & \binom{2n-2}{n-1} & \binom{2n-3}{n-1} & \dots & \binom{n}{n-1} & \binom{n-1}{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \binom{n+1}{1} & \binom{n}{1} & \binom{n-1}{1} & \dots & \binom{2}{1} & \binom{1}{1} \\ \binom{n}{0} & \binom{n-1}{0} & \binom{n-2}{0} & \dots & \binom{1}{0} & \binom{0}{0} \end{bmatrix}. \tag{3.125}$$

Applying elementary row operation to Λ_{n+1} by multiplying the matrix E on the left and elementary column operation to $E\Lambda_{n+1}$ by multiplying the matrix E^T on the right where

$$E = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -1 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}, \tag{3.126}$$

we obtain the following relation:

$$E\Lambda_{n+1}E^T = \begin{bmatrix} \binom{2n-2}{n-1} & \binom{2n-3}{n-1} & \binom{2n-4}{n-1} & \dots & \binom{n-1}{n-1} & 0 \\ \binom{2n-3}{n-2} & \binom{2n-4}{n-2} & \binom{2n-5}{n-2} & \dots & \binom{n-2}{n-2} & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \binom{n-1}{0} & \binom{n-2}{0} & \binom{n-3}{0} & \dots & \binom{0}{0} & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} = \begin{bmatrix} \Lambda_n & O_{n,1} \\ O_{1,n} & 1 \end{bmatrix}. \tag{3.127}$$

Hence Λ_{n+1} is invertible by the induction hypothesis. $\square$

Corollary 3.24. *The following matrix $\Lambda_{n,k} \in \mathbb{R}^{k\times n}$ is full-rank.*

$$\Lambda_{n,k} = \left\{ \binom{2n-i-j}{n-i} \right\}_{n-k+1\leq i\leq n, 1\leq j\leq n}. \tag{3.128}$$

We will use the matrix $\Lambda_{n,k}$ in the proof of Lemma 3.10 to approximate a sequence-to-sequence function.

3.6.3 Proof of Lemma 3.10

Define token-wise lifting map $\mathcal{P} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x+1}$ and modified TRNN cell $\mathcal{TR}_l$: $\mathbb{R}^{(d_x+1)\times N} \to \mathbb{R}^{(d_x+1)\times N}$ as in the proof of Lemma 3.5:

$$\mathcal{P}(x)[t] = \begin{bmatrix} x[m] \\ 0 \end{bmatrix}, \tag{3.129}$$

and

$$\mathcal{TR}_l(X)[t+1] = A_l \mathcal{TR}_l(X)[t] + B_l[t](X)[t+1], \tag{3.130}$$

where $A_l = \begin{bmatrix} O_{d_x,d_x} & O_{d_x,1} \\ O_{1,d_x} & 1 \end{bmatrix}$, $B_l[t] = \begin{bmatrix} I_{d_x} & O_{d_x,1} \\ b_l[t] & 1 \end{bmatrix}$ for $b_l[t] \in \mathbb{R}^{1\times d_x}$. Then we have

$$\begin{aligned} T(n,m) &\coloneqq \mathcal{N}_n(x)[m] \\ &= \begin{bmatrix} x[m] \\ \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} \binom{n+m-i-j}{n-i} b_i[j]x[j] \end{bmatrix}, \end{aligned} \tag{3.131}$$

where $x \in \mathbb{R}^{d_x\times N}$ and $\mathcal{N}_L = \mathcal{TR}_L \circ \mathcal{TR}_{L-1} \circ \cdots \circ \mathcal{TR}_1 \circ \mathcal{P}$.

Since for each t, the matrix

$$\Lambda_{N,N-t+1} = \left\{\binom{2N-i-j}{N-i}\right\}_{t\le i\le N, 1\le j\le N} \tag{3.132}$$

$$= \left\{\binom{2N-t+1-i-j}{N-j}\right\}_{1\le i\le N-t+1, 1\le j\le N}. \tag{3.133}$$

is full-rank, there exist $b_1[t], b_2[t], \ldots, b_N[t]$ satisfying

$$\Lambda_{N,N-t+1}\begin{bmatrix} b_1[t] \\ \vdots \\ b_N[t] \end{bmatrix} = \begin{bmatrix} A_t[N] \\ \vdots \\ A_t[t] \end{bmatrix}, \tag{3.134}$$

or

$$\sum_{j=1}^{N}\binom{N+k-j-t}{N-j} b_j[t] = A_t[k], \tag{3.135}$$

for each $k = 1, 2, \ldots, N$. Then we obtain

$$\begin{aligned} T(N,t) &= \sum_{i=1}^{\infty}\sum_{j=1}^{\infty}\binom{N+t-i-j}{N-i} b_i[j]x[j] \\ &= \sum_{j=1}^{t}\sum_{i=1}^{N}\binom{N+t-i-j}{N-i} b_i[j]x[j] \\ &= \sum_{j=1}^{t} A_j[t]x[j]. \end{aligned} \tag{3.136}$$

3.6.4 Proof of Lemma 3.12

As one of the modified TRNNs that computes (3.33), we use the modified TRNN defined in Appendix 3.6.3. Specifically, we show that for a given l, there exists a modified RNN of width $d_x + 2 + \gamma(\sigma)$ that approximates the modified TRNN cell $\mathcal{TR}_l : \mathbb{R}^{(d_x+1)\times N} \to \mathbb{R}^{(d_x+1)\times N}$ defined by (3.129). Suppose $K \subset \mathbb{R}^{d_x}$, $K' \subset \mathbb{R}$ are compact sets and $X \in (K \times K')^N \subset \mathbb{R}^{(d_x+1)\times N}$. Then the output of the TRNN cell $\mathcal{TR}_l$ is

$$\mathcal{TR}_l(X)[t] = \begin{bmatrix} X[t]_{1:d_x} \\ \sum_{j=1}^{t} b_l[j] X[j]_{1:d_x} + \sum_{j=1}^{t} X[j]_{d_x+1} \end{bmatrix}. \tag{3.137}$$

Without loss of generality, assume $K \subset \left[0, \frac{1}{2}\right]^{d_x}$ and let $\gamma = \gamma(\sigma)$. Let $\mathcal{P} : \mathbb{R}^{d_x+1} \to \mathbb{R}^{d_x+2+\gamma}$ be a token-wise linear map defined by $\mathcal{P}(X) = \begin{bmatrix} X_{1:d_x} \\ 0 \\ X_{d_x+1} \\ \mathbf{0}_\gamma \end{bmatrix}$. Construct the modified recurrent cells $\mathcal{R}_1, \mathcal{R}_2 : \mathbb{R}^{(d_x+2+\gamma(\sigma))\times N} \to \mathbb{R}^{(d_x+2+\gamma(\sigma))\times N}$ as for $X' \in \mathbb{R}^{(d_x+2+\gamma)\times N}$,

$$\mathcal{R}_1(X')[t+1] \tag{3.138}$$

$$= \begin{bmatrix} O_{d_x,d_x} & & \\ & 1 & \\ & & O_{1+\gamma,1+\gamma} \end{bmatrix} \mathcal{R}_1(X')[t] + X'[t+1] + \begin{bmatrix} \mathbf{0}_{d_x} \\ 1 \\ \mathbf{0}_{1+\gamma} \end{bmatrix}, \tag{3.139}$$

and

$$\mathcal{R}_2(X')[t+1] = \begin{bmatrix} I_{d_x} & \mathbf{1}_{d_x} & & \\ & & 1 & \\ & & & O_{1+\gamma,\gamma} \end{bmatrix} X'[t]. \tag{3.140}$$

Then, by definition for $X \in (K \times K')^N$,

$$\mathcal{R}_2\mathcal{R}_1\mathcal{P}(X)[t] = \begin{bmatrix} X[t]_{1:d_x} + t\mathbf{1}_{d_x} \\ t \\ X[t]_{d_x+1} \\ \mathbf{0}_\gamma \end{bmatrix}. \tag{3.141}$$

Note that $D_i = \{\mathcal{R}_2\mathcal{R}_1\mathcal{P}(X)[i]_{1:d_x} \mid X \in (K \times K')^N\} = \{X[i]_{1:d_x} + t\mathbf{1}_{d_x} \mid X \in (K \times K')^N\}$ are disjoint each other, $D_i \cap D_j = \phi$ for all $i \neq j$.

By the universal approximation theorem of deep MLP from [14, 22], for any $\delta_l > 0$, there exists an MLP $\mathcal{N}_{l,MLP} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x+1}$ of width $d_x + 1 + \gamma$ such that for $v \in \mathbb{R}^{d_x}$,

$$\mathcal{N}_{l,MLP}(v)_{1:d_x} = v, \tag{3.142}$$

and

$$\sup_{t=1,\ldots,N} \sup_{v \in D_t} \|b_l[t]\,(v - t\mathbf{1}d_x) - \mathcal{N}_{l,MLP}(v)_{d_x+1}\| < \delta_l. \tag{3.143}$$

Since token-wise MLP is implemented by RNN with the same width, there exists an RNN $\mathcal{N}_l : \mathbb{R}^{d_x+2+\gamma} \to \mathbb{R}^{d_x+2+\gamma}$ of width $d_x + 2 + \gamma$ whose components all but $(d_x + 2)$-th construct $\mathcal{N}_{l,MLP}$ so that for all $X' \in \mathbb{R}^{d_x+2+\gamma}$,

$$\mathcal{N}_l(X')[t] = \begin{bmatrix} \mathcal{N}_{l,MLP}\left(X'[t]_{1:d_x}\right) \\ X'[t]_{d_x+2} \\ \mathbf{0}_\gamma \end{bmatrix}. \tag{3.144}$$

Then for $X \in (K \times K')^N$ we have

$$\mathcal{N}_l \mathcal{R}_2 \mathcal{R}_1 \mathcal{P}(X)[t] = \begin{bmatrix} \mathcal{N}_{l,MLP}\left(X[t]_{1:d_x} + t\mathbf{1}_{d_x}\right) \\ X[t]_{d_x+1} \\ \mathbf{0}_\gamma \end{bmatrix}. \tag{3.145}$$

Finally, define a recurrent cell $\mathcal{R}_3 : \mathbb{R}^{d_x+2+\gamma} \to \mathbb{R}^{d_x+2+\gamma}$ of width $d_x + 2 + \gamma$ as

$$\begin{aligned} &\mathcal{R}_3(X')[t+1] \\ &= \begin{bmatrix} O_{d_x+1,d_x+1} & & \\ & 1 & \\ & & O_{\gamma,\gamma} \end{bmatrix} \mathcal{R}_3(X')[t] + \begin{bmatrix} I_{d_x} & & & \\ & 1 & & \\ & 1 & 1 & \\ & & & O_{\gamma,\gamma} \end{bmatrix} X'[t+1], \end{aligned} \tag{3.146}$$

and attain

$$\begin{aligned} &\mathcal{R}_3 \mathcal{N}_1 \mathcal{R}_2 \mathcal{R}_1 \mathcal{P}(X)[t] \\ &= \begin{bmatrix} X[t]_{1:d_x} + t\mathbf{1}_{d_x} \\ \mathcal{N}_{l,MLP}\left(X[t]_{1:d_x} + t\mathbf{1}_{d_x}\right)_{d_x+1} \\ \sum_{j=1}^{t} \mathcal{N}_{l,MLP}\left(X[j]_{1:d_x} + j\mathbf{1}_{d_x}\right)_{d_x+1} + \sum_{j=1}^{t} X[j]_{d_x+1} \\ \mathbf{0}_\gamma \end{bmatrix}. \end{aligned} \tag{3.147}$$

With the token-wise projection map $\mathcal{Q} : \mathbb{R}^{d_x+2+\gamma} \to \mathbb{R}^{d_x+1}$ defined by $\mathcal{Q}(X') = \begin{bmatrix} X'_{1:d_x} \\ X'_{d_x+2} \end{bmatrix}$, an RNN $\mathcal{Q}\mathcal{R}_3\mathcal{N}_l\mathcal{R}_2\mathcal{R}_1\mathcal{P} : \mathbb{R}^{(d_x+1)\times N} \to \mathbb{R}^{(d_x+1)\times N}$ of width $d_x + 2 + \gamma$

maps $X \in \mathbb{R}^{(d_x+1)\times N}$ to

$$\begin{aligned}
&\mathcal{Q}\mathcal{R}_3\mathcal{N}_l\mathcal{R}_2\mathcal{R}_1\mathcal{P}(X)[t] \\
&\qquad = \begin{bmatrix} X[t]_{1:d_x} + t\mathbf{1}_{d_x} \\ \sum_{j=1}^{t} \mathcal{N}_{l,MLP}\left(X[j]_{1:d_x} + j\mathbf{1}_{d_x}\right)_{d_x+1} + \sum_{j=1}^{t} X[j]_{d_x+1} \end{bmatrix}.
\end{aligned} \tag{3.148}$$

Since $\mathcal{N}_{l,MLP}\left(X[j]_{1:d_x} + j\mathbf{1}_{d_x}\right)_{d_x+1} \to b_l[j]X[j]_{1:d_x}$, we have

$$\sup_{X\in(K\times K')^N} \|\mathcal{T}\mathcal{R}_l(X) - \mathcal{Q}\mathcal{R}_3\mathcal{N}_l\mathcal{R}_2\mathcal{R}_1\mathcal{P}(X)\| \to 0, \tag{3.149}$$

as $\delta_l \to 0$. Approximating all $\mathcal{T}\mathcal{R}_l$ in Appendix 3.6.3 finishes the proof.

3.6.5 Proof of Lemma 3.18

The main idea of the proof is to separate the linear sum $\sum_{j=1}^{N} A_j[t]x[j]$ into the past-dependent part $\sum_{j=1}^{t-1} A_j[t]x[j]$ and the remainder part $\sum_{j=t}^{N} A_j[t]x[j]$. Then, we construct modified TBRNN with $2N$ cells; the former N cells have only a forward recurrent cell to compute the past-dependent part, and the latter N cells have only a backward recurrent cell to compute the remainder.

Let the first N modified TRNN cells $\mathcal{R}_l : \mathbb{R}^{(d_x+1)\times N} \to \mathbb{R}^{(d_x+1)\times N}$ for $1 \le l \le N$ be defined as in the proof of Lemma 3.10:

$$\mathcal{R}_l\left(X\right)[t+1] = A_l\mathcal{R}_l\left(X\right)[t] + B_l[t]X[t+1], \tag{3.150}$$

where $A_l = \begin{bmatrix} O_{d_x,d_x} & O_{d_x,1} \\ O_{1,d_x} & 1 \end{bmatrix}$, $B_l[t] = \begin{bmatrix} I_{d_x} & O_{d_x,1} \\ b_l[t] & 1 \end{bmatrix}$ for $b_l[t] \in \mathbb{R}^{1\times d_x}$. Then, with token-wise lifting map $\mathcal{P} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x+1}$ defined by $\mathcal{P}(x) = \begin{bmatrix} x \\ 0 \end{bmatrix}$, we construct modified TRNN $\mathcal{N} : \mathcal{R}_N \circ \cdots \circ \mathcal{R}_1 \circ \mathcal{P} : \mathbb{R}^{d_x \times N} \to \mathbb{R}^{(d_x+1)\times N}$. We know that if $C_i[m] \in \mathbb{R}^{1\times d_x}$ are given for $1 \leq m \leq N$ and $1 \leq i \leq m$, there exist $b_l[t]$ for $1 \leq l \leq N$, such that

$$\mathcal{N}_N(x)[m] = \begin{bmatrix} x[m] \\ \sum_{i=1}^{m} C_i[m]x[i] \end{bmatrix}. \tag{3.151}$$

Therefore, we will determine $C_i[m]$ after constructing the latter N cells. Let $f_m = \sum_{i=1}^{m} C_i[m]x[i]$ for brief notation.

After $\mathcal{N}_N$, construct N modified TRNN cells $\bar{\mathcal{R}}_l : \mathbb{R}^{(d_x+1)\times N} \to \mathbb{R}^{(d_x+1)\times N}$ for $1 \leq l \leq N$ in reverse order:

$$\bar{\mathcal{R}}_l\left(\bar{X}\right)[t-1] = \bar{A}_l \bar{\mathcal{R}}_l\left(\bar{X}\right)[t] + \bar{B}_l[t]\bar{X}[t-1], \tag{3.152}$$

where $\bar{A}_l = \begin{bmatrix} O_{d_x,d_x} & O_{d_x,1} \\ O_{1,d_x} & 1 \end{bmatrix}$, $\bar{B}_l[t] = \begin{bmatrix} I_{d_x} & O_{d_x,1} \\ \bar{b}_l[t] & 1 \end{bmatrix}$ for $\bar{b}_l[t] \in \mathbb{R}^{1\times d_x}$. Define $\bar{\mathcal{N}}_N = \bar{\mathcal{R}}_N \circ \cdots \circ \bar{\mathcal{R}}_1$, and we obtain the following result after a similar calculation with input sequence $\bar{X}[t] = \mathcal{N}_N(x)[t] = \begin{bmatrix} x[t] \\ f_t \end{bmatrix}$:

$$\bar{\mathcal{N}}_N\left(\bar{X}\right)[N+1-t] = \begin{bmatrix} x[N+1-t] \\ Z \end{bmatrix}, \tag{3.153}$$

where $Z = \sum_{j=1}^{t}\left[\sum_{i=1}^{N}\binom{N+t-i-j}{N-i}\bar{b}_i[N+1-j]x[N+1-j] + \binom{N+t-1-j}{N-1}f_{N+1-j}\right]$.

We want to find f_m and $\bar{b}_i[m]$ so that

$$\bar{\mathcal{N}}_N\left(\bar{X}\right)[N+1-t]_{d_x+1} = \sum_{i=1}^{N} A_i[N+1-t]x[i], \tag{3.154}$$

for each $t = 1, 2, \ldots, N$.

Note that $\sum_{j=1}^{t}\sum_{i=1}^{N}\binom{N+t-i-j}{N-i}\bar{b}_i[N+1-j]x[N+1-j]$ does not contain $x[1], x[2], \ldots, x[N-t]$ terms, so $\sum_{j=1}^{t}\binom{N+t-1-j}{N-1}f_{N+1-j}$ should contain $\sum_{i=1}^{N-t}A_i[N+1-t]x[i]$.

$$\begin{aligned}
&\sum_{j=1}^{t}\binom{N+t-1-j}{N-1}f_{N+1-j} && (3.155)\\
&\quad=\sum_{j=1}^{t}\binom{N+t-1-j}{N-1}\sum_{i=1}^{N+1-j}C_i[N+1-j]x[i] && (3.156)\\
&\quad=\sum_{j=1}^{t}\sum_{i=1}^{N+1-j}\binom{N+t-1-j}{N-1}C_i[N+1-j]x[i] && (3.157)\\
&\quad=\sum_{i=N+2-t}^{N}\sum_{j=1}^{N+1-i}\binom{N+t-1-j}{N-1}C_i[N+1-j]x[i] && (3.158)\\
&\qquad+\sum_{i=1}^{N+1-t}\sum_{j=1}^{t}\binom{N+t-1-j}{N-1}C_i[N+1-j]x[i]. && (3.159)
\end{aligned}$$

Since matrix $\Lambda_i = \left\{\binom{N+t-1-j}{N-1}\right\}_{1\leq t\leq N+1-i, 1\leq j\leq N+1-i}$ is a lower triangular $(N+1-i)\times(N+1-i)$ matrix with unit diagonal components, there exist $C_i[i], C_i[i+1], \ldots, C_i[N]$ such that

$$\sum_{j=1}^{t}\binom{N+t-1-j}{N-1}C_i[N+1-j] = A_i[N+1-t], \tag{3.160}$$

for each $t = 1, 2, \ldots, N+1-i$.

We now have

$$\sum_{j=1}^{t} \binom{N+t-1-j}{N-1} f_{N+1-j} \tag{3.161}$$

$$= \sum_{i=N+2-t}^{N} \sum_{j=1}^{N+1-i} \binom{N+t-1-j}{N-1} C_i[N+1-j]x[i] \tag{3.162}$$

$$+ \sum_{i=1}^{N+1-t} A_i[N+1-t]x[i] \tag{3.163}$$

$$= \sum_{i=1}^{t-1} \sum_{j=1}^{i} \binom{N+t-1-j}{N-1} C_{N+1-i}[N+1-j]x[N+1-i] \tag{3.164}$$

$$+ \sum_{i=1}^{N+1-t} A_i[N+1-t]x[i] \tag{3.165}$$

$$= \sum_{j=1}^{t-1} \sum_{i=1}^{j} \binom{N+t-1-i}{N-1} C_{N+1-j}[N+1-i]x[N+1-j] \tag{3.166}$$

$$+ \sum_{i=1}^{N+1-t} A_i[N+1-t]x[i]. \tag{3.167}$$

We switch i and j for the last equation. By Corollary 3.24, there exist $\bar{b}_i[N+1-j]$ satisfying

$$\sum_{i=1}^{N} \binom{N+t-i-j}{N-i} \bar{b}_i[N+1-j] \tag{3.168}$$

$$= A_{N+1-j}[N+1-t] - \sum_{i=1}^{j} \binom{N+t-1-i}{N-1} C_{N+1-j}[N+1-i], \tag{3.169}$$

for $j = 1, 2, \ldots, t-1$, and

$$\sum_{i=1}^{N} \binom{N+t-i-j}{N-i} \bar{b}_i[N+1-j] = A_{N+1-j}[N+1-t], \tag{3.170}$$

for $j = t$.

With the above $C_i[m]$ and $\bar{b}_i[m]$, equation (3.154) holds for each $t = 1, 2, \ldots, N$. It remains to construct modified TRNN cells to implement f_m, which comes directly from the proof of Lemma 3.10.

Chapter 4

Conclusion

In this thesis, we investigated the universality of the recurrent neural network and the convolutional neural network.

In Chapter 2, we studied the universality of convolutional neural networks with both limited depth and unlimited width and with limited width and unlimited depth. Although we have only dealt with the universality of three-kernel convolutions, we expect that the same idea can be simply generalized to networks of other kernel sizes. We think that convolution using striding and dilation and the convolutional layer mixed with pooling are also interesting research topics for the universality of convolutional neural networks. We hope that our research will serve as a basis for active research in this field.

In Chapter 3, we investigated the universality and upper bound of the minimum width of deep RNNs. The upper bound of the minimum width serves as a theoretical basis for the effectiveness of deep RNNs, especially when underlying dynamics of the data are unknown.

Our methodology enables various follow-up studies, as it connects an MLP and a deep RNN. For example, the framework disentangles the time dependency of output sequence of an RNN. This makes it feasible to investigate a trade-off between width and depth in the representation ability or error bounds of the deep RNN, which has not been studied because of the entangled flow with time and depth. In addition, we separated the required width into three parts: one maintains inputs and results, another resolves the time dependency, and the third modifies the activation. Assuming some underlying dynamics in the output sequence, such as an open dynamical system, we expect to reduce the required minimum width on each part because there is a natural dependency between the outputs, and the inputs are embedded in a specific way by the dynamics.

However, as LSTMs and GRUs have multiple hidden states in the cell process, they may have a smaller minimum width than the RNN. By constructing an LSTM and a GRU to use the hidden states to save data and resolve the time dependency, we hope that our techniques demonstrated in the proof help analyze why these networks have a better result in practice and suffer less from long-term dependency.

Clarification

This thesis was written by revising and combining some of the author's works, [19] and [38].

Bibliography

[1] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, *wav2vec 2.0: A framework for self-supervised learning of speech representations*, Advances in Neural Information Processing Systems, 33 (2020), pp. 12449–12460.

[2] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473, (2014).

[3] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, *End-to-end attention-based large vocabulary speech recognition*, in 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, 2016, pp. 4945–4949.

[4] A. Bhoi, *Monocular depth estimation: A survey*, arXiv preprint arXiv:1901.09402, (2019).

[5] N. Cohen, O. Sharir, and A. Shashua, *On the expressive power of deep learning: A tensor analysis*, in Conference on Learning Theory, PMLR, 2016, pp. 698–728.

[6] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems, 2 (1989), pp. 303–314.

[7] Z. Dai, H. Liu, Q. V. Le, and M. Tan, *Coatnet: Marrying convolution and attention for all data sizes*, Advances in Neural Information Processing Systems, 34 (2021), pp. 3965–3977.

[8] L. De Branges, *The stone-weierstrass theorem*, Proceedings of the American Mathematical Society, 10 (1959), pp. 822–824.

[9] J. L. Elman, *Finding structure in time*, Cognitive science, 14 (1990), pp. 179–211.

[10] A. A. Elngar, M. Arafa, A. Fathy, B. Moustafa, O. Mahmoudm, M. Shaban, and N. Fawzy, *Image classification based on cnn: a survey*, J. Cybersecurity Inf. Manag.(JCIM), 6 (2021), pp. 18–50.

[11] L. Fan, F. Zhang, H. Fan, and C. Zhang, *Brief review of image denoising techniques*, Visual Computing for Industry, Biomedicine, and Art, 2 (2019), pp. 1–12.

[12] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami, and Y. W. Teh, *Neural processes*, arXiv preprint arXiv:1807.01622, (2018).

[13] A. Graves and N. Jaitly, *Towards end-to-end speech recognition with recurrent neural networks*, in International Conference on Machine Learning, PMLR, 2014, pp. 1764–1772.

[14] B. Hanin and M. Sellke, *Approximating continuous functions by relu nets of minimal width*, arXiv preprint arXiv:1710.11278, (2017).

[15] J. Hanson and M. Raginsky, *Universal simulation of stable dynamical systems by recurrent neural nets*, in Learning for Dynamics and Control, PMLR, 2020, pp. 384–392.

[16] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[17] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, *Session-based recommendations with recurrent neural networks*, arXiv preprint arXiv:1511.06939, (2015).

[18] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural networks, 2 (1989), pp. 359–366.

[19] G. Hwang and M. Kang, *Universal property of convolutional neural networks*, arXiv preprint arXiv:2211.09983, (2022).

[20] J. Johnson, *Deep, skinny neural networks are not universal approximators*, in International Conference on Learning Representations, 2019.

[21] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, *Exploring the limits of language modeling*, arXiv preprint arXiv:1602.02410, (2016).

[22] P. Kidger and T. Lyons, *Universal approximation with deep narrow networks*, in Conference on learning theory, PMLR, 2020, pp. 2306–2327.

[23] P. Lavanya and E. Sasikala, *Deep learning techniques on text classification using natural language processing (nlp) in social healthcare network: A com-*

prehensive survey, in 2021 3rd International Conference on Signal Processing and Communication (ICPSC), IEEE, 2021, pp. 603–609.

[24] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, nature, 521 (2015), pp. 436–444.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[26] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*, Neural networks, 6 (1993), pp. 861–867.

[27] H. Lin and S. Jegelka, *Resnet with one-neuron hidden layers is a universal approximator*, Advances in Neural Information Processing Systems, 31 (2018).

[28] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[29] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, *The expressive power of neural networks: A view from the width*, in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017.

[30] H. Maron, E. Fetaya, N. Segol, and Y. Lipman, *On the universality of invariant networks*, in International conference on machine learning, PMLR, 2019, pp. 4363–4371.

[31] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, *Recurrent neural network based language model.*, in Interspeech, vol. 2, Makuhari, 2010, pp. 1045–1048.

[32] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, arXiv preprint arXiv:1511.08458, (2015).

[33] S. Park, C. Yun, J. Lee, and J. Shin, *Minimum width for universal approximation*, in International Conference on Learning Representations, 2021.

[34] D. Rolnick and M. Tegmark, *The power of deeper networks for expressing natural functions*, In the International Conference on Learning Representations, (2018).

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, nature, 323 (1986), pp. 533–536.

[36] A. M. Schäfer and H.-G. Zimmermann, *Recurrent neural networks are universal approximators*, International journal of neural systems, 17 (2007), pp. 253–263.

[37] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).

[38] C. Song, G. Hwang, and M. Kang, *Minimal width for universal property of deep rnn*, arXiv preprint arXiv:2211.13866, (2022).

[39] R. Suthar and M. K. R. Patel, *A survey on various image inpainting techniques to restore image*, Int. Journal of Engineering Research and Applications, 4 (2014), pp. 85–88.

[40] X. Tan, T. Qin, F. Soong, and T.-Y. Liu, *A survey on neural speech synthesis*, arXiv preprint arXiv:2106.15561, (2021).

[41] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, *Recurrent recommender networks*, in Proceedings of the tenth ACM international conference on web search and data mining, 2017, pp. 495–503.

[42] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492–1500.

[43] D. Yarotsky, *Universal approximations of invariant maps by neural networks*, Constructive Approximation, 55 (2022), pp. 407–474.

[44] C. Yun, S. Bhojanapalli, A. S. Rawat, S. Reddi, and S. Kumar, *Are transformers universal approximators of sequence-to-sequence functions?*, in International Conference on Learning Representations, 2020.

[45] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, *A survey of modern deep learning based object detection models*, Digital Signal Processing, (2022), p. 103514.

[46] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, *Scaling vision transformers*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12104–12113.

[47] K. Zhang, W. Ren, W. Luo, W.-S. Lai, B. Stenger, M.-H. Yang, and H. Li, *Deep image deblurring: A survey*, International Journal of Computer Vision, 130 (2022), pp. 2103–2130.

[48] D.-X. Zhou, *Universality of deep convolutional neural networks*, Applied and computational harmonic analysis, 48 (2020), pp. 787–794.

국문초록

특정 함수 공간의 임의의 함수를 함수 집합이 근사할 수 있는지 여부를 의미하는 보편 근사 가능성을 판별하는 것은 뉴럴 네트워크의 큰 발전에 힘입어 최근 활발히 연구 되고 있다. 뉴럴 네트워크는 다양한 구조에 따라 함수에 다양한 제약 조건을 발생 시키고 근사할 수 있는 함수의 범위가 달라지게 되며, 다른 함수 공간을 목적으로 하면 그 목적에 대응하는 보편 근사 정리가 필요하게 된다. 이런 목적에 맞춰 본 논문에서 우리는 합성곱 신경망과 순환 신경망, 두가지 서로 다른 딥러닝 네트워크 구조에 대한 보편 근사 정리를 증명하였다.

첫째로 우리는 합성곱 신경망의 보편성에 대해 증명하였다. 패딩이 적용된 합성곱은 입력값과 동일한 형태의 값을 출력하게 되며 이에 따라 합성곱으로 구성된 합성곱 신경망이 이와 같은 함수를 근사가능한지 여부를 증명할 필요가 있다. 우리는 입력값과 출력값이 동일한 형태를 가지는 연속 함수에 대하여 합성곱 신경망이 보편적으로 근사가능하다는 것을 증명하였다. 또한 근사에 필요한 신경망의 최소 깊이를 제시하였으며 이것이 최적 값임을 증명하였다. 또한 채널의 개수가 제한된 상황에서 충분히 깊은 층을 가지는 합성곱 신경망이 마찬가지로 보편성을 가진다는 것을 증명하였다.

둘째로 우리는 순환 신경망의 보편성을 증명하였다. 순환 신경망은 시간 순서의 앞부분에 위치한 입력값에 의해 뒷부분의 출력값이 결정되는 과거 의존성을 가지며 우리는 순환 신경망의 과거 의존적 함수 공간에서의 보편성에 대해 연구하였다. 구체적으로 우리는 채널의 개수가 제한된 다층 순환 신경망이 임의의 연속함수와 L_p 함수를 각각 근사할 수 있다는 것을 증명하였다. 또한 양방향 순환신경망과 GRU, LSTM에도 본 결과를 확장하였다.

주요어휘: 보편 근사 정리, 순환 신경망, 합성곱 신경망, 심층 협소 신경망

학번: 2017-25155