Journal of Machine Learning Research 23 (2022) 1-50 Submitted 10/21; Revised 7/22; Published 10/22

# Toward Understanding Convolutional Neural Networks from Volterra Convolution Perspective

**Tenghui Li** TENGHUI.LEE@FOXMAIL.COM
*School of Automation, Guangdong University of Technology, Guangzhou, China*
*Key Laboratory of Intelligent Information Processing and System Integration of IoT, Ministry of Education, Guangdong University of Technology, Guangzhou, China*

**Guoxu Zhou** * GX.ZHOU@GDUT.EDU.CN
*School of Automation, Guangdong University of Technology, Guangzhou, China*
*Key Laboratory of Intelligent Detection and The IoT in Manufacturing, Ministry of Education, Guangdong University of Technology, Guangzhou, China*

**Yuning Qiu** YUNING.QIU.GD@GMAIL.COM
*Guangdong Key Laboratory of IoT Information Technology, Guangdong University of Technology, Guangzhou, China*
*Guangdong-Hong Kong-Macao Joint Laboratory for Smart Discrete Manufacturing, Guangdong University of Technology, Guangzhou, China*

**Qibin Zhao** QIBIN.ZHAO@RIKEN.JP
*RIKEN Center for Advanced Intelligence Project, Tokyo, Japan*
*School of Automation, Guangdong University of Technology, Guangzhou, China*

**Editor:** Joan Bruna

## Abstract

We make an attempt to understand convolutional neural network by exploring the relationship between (deep) convolutional neural networks and Volterra convolutions. We propose a novel approach to explain and study the overall characteristics of neural networks without being disturbed by the horribly complex architectures. Specifically, we attempt to convert the basic structures of a convolutional neural network (CNN) and their combinations to the form of Volterra convolutions. The results show that most of convolutional neural networks can be approximated in the form of Volterra convolution, where the approximated proxy kernels preserve the characteristics of the original network. Analyzing these proxy kernels may give valuable insight about the original network. Based on this setup, we present methods to approximate the order-zero and order-one proxy kernels, and verify the correctness and effectiveness of our results.

**Keywords:** Convolutional neural network, Volterra convolution, order-n convolution, unified neural network, proxy kernel

∗. Corresponding author.

## 1. Introduction

Deep neural networks (DNNs) can effectively characterize most complex data as long as the training data is large enough and the capable models are well-trained. Nevertheless, a deep neural network often has a horribly complex structure, the results are hard to interpret in some sense, and the neural network is likely to be deceived by adversarial examples. We are eager to search for methods that allow us to analyze the neural network.

There is a vast number of excellent researches focusing on theoretically understanding neural networks. Some take the statistical perspective. Deep neural networks can be thought of as being discrete dynamical systems (E, 2017). Instead of thinking about features and neurons, one focus on representation of functions, calculus of variation problems, and continuous gradient flow (E et al., 2020). Besides, others take a geometric perspective (grids, groups, graphs, geodesics, and gauges), which shows that deep neural networks can be understood in a unified manner as methods that respect the structure and symmetries (invariants and equivalents) of the geometric domains (Bronstein et al., 2021). Moreover, we can study how modern deep neural networks transform topologies of data sets (Naitzat et al., 2020), or draw the phase diagram for the two-layer ReLU neural network at the infinite-width limit (Luo et al., 2021).

Most of these works focus on certain classes of structures, such as two-layer neural network, multilayer fully connected network, ResNet, pure abstract network, and fully connected network with specific activation functions, i.e., ReLU, sigmoid, tanh, and so on. It seems that it is unlikely to represent and analyze an arbitrarily complex neural network from a theoretical point of view.

In this paper, we attempt to build a generic and unified model for analyzing most deep convolutional neural networks rather than thinking about features and layers. The neural network is a universal approximator that is able to approximate any Borel measurable function (Cybenko, 1989; Hornik et al., 1989; Barron, 1993; Zeng et al., 2021). The proposed model is expected to be a universal approximator. Additionally, it is supposed to ensure that most networks can be represented by this kind of model. Furthermore, it should be expressed as superposition of submodules, which makes it convenient to be analyzed.

The Volterra convolution or Volterra series operator (Volterra, 1932) owns exactly such features. Briefly, Volterra convolution has the form of

$$\mathbf{y} = \mathbf{H}_0 + \mathbf{H}_1 * \mathbf{x} + \mathbf{H}_2 * \mathbf{x}^2 + \cdots = \sum_{n=0}^{+\infty} \mathbf{H}_n * \mathbf{x}^n, \tag{1}$$

where $\mathbf{x}$ is the input signal, $\mathbf{H}_0, \mathbf{H}_1, \mathbf{H}_2, \cdots$ are kernels, $\mathbf{y}$ is the output signal, and $\mathbf{H}_n * \mathbf{x}^n$ is the order-$n$ convolution (All of these will be precisely defined in Section 2).

Firstly, it has been proved that any time-invariant continuous nonlinear operator can be approximated by a Volterra series operator and any time invariant operator with fading memory can be approximated (in a strong sense) by a nonlinear moving-average operator, the finite term Volterra series operator (Boyd and Chua, 1985). Secondly, a certain class of artificial neural networks (feed-forward network or multilayer perceptron) are equivalent to a finite memory Volterra series (Wray and Green, 1994; Fung et al., 1996). In addition to this certain class of artificial neural networks, we show that neural networks, including convolutional neural networks and their numerous variants, can be approximated in the

form of Volterra convolution. Thirdly, this is an accumulation of multiple submodules $\mathbf{H}_1 * \mathbf{x}, \mathbf{H}_2 * \mathbf{x}^2, \cdots$, which can be analyzed independently and without being disturbed by the complex network architecture.

Suppose the well-trained neural network is $f(\mathbf{x})$ or $g(f(\mathbf{x}))$, we are looking for a Volterra convolution to approximate this network $f(\mathbf{x})$. The functions learned by practical convolutional neural networks (i.e., ReLU-based, sigmoid-based) often can only be represented by Volterra convolution with *infinite* series. Nevertheless, if small truncation errors are allowed in practice, we can use *finite* term Volterra convolution via truncating the infinite counterpart to approximate the functions, which is mainly considered in this paper. Formally, for a function $f(\mathbf{x})$, we are looking for $N+1$ proxy kernels $\mathbf{H}_0, \mathbf{H}_1, \cdots, \mathbf{H}_N$ such that

$$f(\mathbf{x}) \approx \sum_{n=0}^{N} \mathbf{H}_n * \mathbf{x}^n. \tag{2}$$

If a neural network can be approximated in this form, its kernels $\mathbf{H}_0, \mathbf{H}_1, \cdots, \mathbf{H}_N$ shall preserve characteristics of the original network, and they will probably help us to analyze the stability or robustness or other useful properties of a well-trained network.

The Volterra convolution looks like a polynomial network (Giles and Maxwell, 1987; Shin and Ghosh, 1995, 1992; Fallahnezhad et al., 2011). They are similar in the sense of "polynomial". Nevertheless, they are quite different. A polynomial network learns a function $f : \mathbb{R}^n \to \mathbb{R}$, but Volterra convolution is a map $f : \mathbb{R}^n \to \mathbb{R}^m$. Besides, we are not interested in training this Volterra convolution by raw data. Instead, we will just approximate a well-trained network in this formulation of Volterra convolution.

In summary, the main contribution of this paper include:

- We show that most convolutional neural networks can be represented in the form of Volterra convolutions. This formulation provides a novel perspective on understanding neural networks, and will probably help us to analyze a neural network without being disturbed by its complex architecture.
- We study some important properties of this representation, including the effects of perturbations and the rank of combining two order-$n$ convolutions.
- We show that a convolutional neural network can be well approximated by finite term Volterra convolutions. All we need in this approximation are the proxy kernels. We propose two methods to infer the proxy kernels. The first is the direct calculation, provided that the original network is a white-box to users, whereas the second one is to train a hacking network, for the case where the original network is a black-box.

The main structure of this article is listed as follows. In Section 2, we introduce and review the definition and properties of order-$n$ convolution, outer convolution and Volterra convolution. In Section 3, we show that most convolutional neural networks can be approximated in the form of Volterra convolutions, and validate this approximation on simple structures. In Section 4, we propose a hacking network to approximate the order-zero and order-one proxy kernels.

*Notations.* In most situations, vectors are notated in lower case bold face letters $\mathbf{x}, \mathbf{y}, \cdots$, while matrices and tensors are notated in upper case bold face letters $\mathbf{H}, \mathbf{G}, \cdots$. We will not

deliberately distinguish between them, as this does not affect the generality of our discussion. Elements are notated by regular letters with brackets $x(i), X(i,j,k), \cdots$. The $i$th-slice of $\mathbf{X}$ is noted by $X(:,i,:)$ using the colon notation.

The vector like notation is $\vec{\mathbf{x}} = \lceil \mathbf{x}_1, \mathbf{x}_2, \cdots \rfloor$, which is nothing but a list of objects and the addition and subtraction are defined as $\vec{\mathbf{x}} \pm l = \lceil x_1 \pm l, x_2 \pm l, \cdots \rfloor$ and $\vec{\mathbf{x}} \pm \vec{\mathbf{y}} = \lceil x_1 \pm y_1, x_2 \pm y_2, \cdots \rfloor$.

## 2. Extension of Convolutions

In this section, four kinds of convolutions are introduced in both continuous and discrete time, including the well-known convolution (Equation 3), order-$n$ convolution (Definition 2), Volterra convolution (Definition 3), and outer convolution (Definition 4).

Without loss of generality, all kernels and signals are bounded by a constant $M_1 < \infty$ and have Lipschitz constant $M_2 < \infty$,

$$\{\mathbf{x} \in C(\mathbb{R}) : |x(t)| \le M_1, |x(s) - x(t)| \le M_2(s-t), \text{ for } t \le s\},$$

where $C(\mathbb{R}) : \mathbb{R} \to \mathbb{R}$ is the space of bounded continuous functions, and all kernels are absolute integrable $\int_{-\infty}^{+\infty} |h(t)| dt < \infty$ or $\sum_t |h(t)| < \infty$.

The well known one-dimensional convolution (Gonzalez and Woods, 2017) of kernel $\mathbf{h}$ and signal $\mathbf{x}$ is

$$\begin{aligned} (\mathbf{h} * \mathbf{x})(t) &= \int_{-\infty}^{+\infty} h(\tau) x(t-\tau) d\tau \quad \text{(continuous)}, \\ (\mathbf{h} * \mathbf{x})(t) &= \sum_{\tau} h(\tau) x(t-\tau) \qquad\quad \text{(discrete)}. \end{aligned} \tag{3}$$

**Remark 1** *If kernel size equals to signal size and padding is zero, the one-dimensional discrete order-one convolution is equivalents to vector inner product with flipped kernel at* $t = 0$*,*

$$\sum_{\tau} h(\tau) x(t-\tau) \Rightarrow \sum_{\tau} h(-\tau) x(\tau).$$

In signal processing, discrete convolution is notated in minus type, such as $\sum_\tau h(\tau)x(t-\tau)$, and correlation is defined in plus type, like $\sum_\tau h(\tau)x(t+\tau)$. While in convolutional networks, we prefer to notate convolution in plus type. These two operations can be converted from one to the other by flipping kernels and shifting time (see Figure 1). Discussing only in minus type does not affect generality of the results.


Figure 1: Differences between addition type and subtraction type.

### 2.1 Order-n Convolution

Nonlinearities of convolutional neural networks come from their nonlinear activation functions. Theoretically, it is possible to embed nonlinearity in convolutional operation by taking order-$n$ convolution.

For a simple example, a curve can be approximated by a two-layer network with activation $\sigma(\cdot)$, i.e., $w_2\sigma(w_1x+b_1)+b_2$, or it can be expressed in polynomial $\alpha_0+\alpha_1x+\alpha_2x^2+\cdots$. This type of "polynomial" can also be applied to convolutional operation. For instance,

$$\begin{aligned}
\alpha x^2 &\leftrightarrows \int_{-\infty}^{+\infty} H(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)d\tau_1 d\tau_2,\\
\alpha xy &\leftrightarrows \int_{-\infty}^{+\infty} H(\tau_1,\tau_2)x(t-\tau_1)y(t-\tau_2)d\tau_1 d\tau_2,\\
\alpha x^3 &\leftrightarrows \int_{-\infty}^{+\infty} H(\tau_1,\tau_2,\tau_3)x(t-\tau_1)x(t-\tau_2)x(t-\tau_3)d\tau_1 d\tau_2 d\tau_3.
\end{aligned}$$

With these basic concepts in mind, formal definition of order-$n$ convolution is presented in Definition 2.

**Definition 2** *Order-n convolution (Volterra, 1932; Rugh, 1981) of kernel* $\mathbf{H}$ *and n signals* $\lceil \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \rfloor \equiv \vec{\mathbf{x}}$ *is*

$$\mathbf{H} * \lceil \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \rfloor \equiv \mathbf{H} * \vec{\mathbf{x}}, \tag{4}$$

*where* $\vec{\mathbf{x}}$ *is vector like notation and all* $\mathbf{x}_i$ *have the same dimension.*

If $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are all one-dimensional signals and $\mathbf{H}$ is an $n$-dimensional signal, the continuous order-$n$ convolution for one-dimensional signal is

$$\begin{aligned}
(\mathbf{H} * \vec{\mathbf{x}})(t) &= (\mathbf{H} * \lceil \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \rfloor)(t)\\
&= \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} H(\tau_1, \tau_2, \cdots, \tau_n) \prod_{i=1}^{n} (x_i(t-\tau_i)d\tau_i)\\
&\equiv \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} H(\vec{\tau}) \prod_{i=1}^{n} (x_i(t-\tau_i)d\tau_i)\,.
\end{aligned} \tag{5}$$

If $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are all $m$-dimensional signals and $\mathbf{H}$ is an $nm$-dimensional signal, and vector like notations have the form $\vec{\tau}_1 = \lceil \tau_{1,1}, \tau_{1,2}, \cdots, \tau_{1,m} \rfloor$; $\cdots$; $\vec{\tau}_n = \lceil \tau_{n,1}, \tau_{n,2}, \cdots, \tau_{n,m} \rfloor$ and $\vec{\mathbf{t}} = \lceil t_1, t_2, \cdots, t_m \rfloor$, order-$n$ convolution for $m$-dimensional signal is

$$\begin{aligned}
(\mathbf{H} * \vec{\mathbf{x}})(t_1, \cdots, t_m) &= (\mathbf{H} * \lceil \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \rfloor)(t_1, \cdots, t_m)\\
&= \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} H(\tau_{1,1}, \cdots, \tau_{1,m}; \cdots; \tau_{n,1}, \cdots, \tau_{n,m})\\
&\qquad \prod_{i=1}^{n} (x_i(t_1-\tau_{i,1}, \cdots, t_m-\tau_{i,m})d\tau_{i,1}\cdots d\tau_{i,m})\\
&\equiv \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} H(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} \left(x(\vec{\mathbf{t}}-\vec{\tau}_i)d\vec{\tau}_i\right).
\end{aligned} \tag{6}$$

With the vector like notation, discrete order-$n$ convolution for $m$-dimensional signal is simplified as

$$(\mathbf{H} * \vec{\mathbf{x}})(\vec{\mathbf{t}}) \equiv \sum_{\vec{\tau}_1,\cdots,\vec{\tau}_n} H(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} x_i(\vec{\mathbf{t}} - \vec{\tau}_i). \tag{7}$$

If $n = 0$, order-zero convolution $\mathbf{H} * \mathbf{x}^0 = \mathbf{H} * \delta = \mathbf{H}$, where $\delta$ is the Dirac delta,

$$\delta(t) = \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases} \text{ (continuous)}, \quad \delta(t) = \begin{cases} 1, & t = 0 \\ 0, & t \neq 0 \end{cases} \text{ (discrete)}.$$

If $n = 1$, this is order-one (first-order) convolution (Equation 3), and $n = 2$, this is order-two (second-order) convolution $\mathbf{H} * \lceil \mathbf{x}, \mathbf{y} \rfloor$. The order-two convolution does not come from void, as it is an extension of order-one convolution. (Please read Appendix A for more details.)

For better understanding this notation, few examples are expressed as follows.

$$\mathbf{H} * \lceil \mathbf{x}, \mathbf{y}, \mathbf{z}, \cdots \rfloor$$

The number of signal equals the order.

Besides, if all signals are equal $\mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_n = \mathbf{x}$, it can be written as $\mathbf{H} * \mathbf{x}^n$ for short.

$$\mathbf{H} * \lceil \mathbf{x}, \mathbf{x}, \cdots, \mathbf{x} \rfloor = \mathbf{H} * \mathbf{x}^n$$

$n$ equals the order.

Dimension of this convolution is determined by its signals. If $\mathbf{x}_j, j = 1, 2, \cdots, n$ are all $m$-dimensional signals, $\mathbf{H} * \vec{\mathbf{x}}$ is called order-$n$ convolution for $m$-dimensional signal.

$$\mathbf{H} * \lceil \cdots, [x_j(i_1, i_2, \cdots, i_m)], \cdots \rfloor$$

$m$ equals the dimension of signal.

If signals are grouped and $n_1 + n_2 + \cdots + n_m$ equals the order, it can simplify as

$$\mathbf{H} * \left\lceil \underbrace{\mathbf{x}_1, \cdots, \mathbf{x}_1}_{n_1 \text{ terms}}, \underbrace{\mathbf{x}_2, \cdots, \mathbf{x}_2}_{n_2 \text{ terms}}, \cdots, \underbrace{\mathbf{x}_m, \cdots, \mathbf{x}_m}_{n_m \text{ terms}} \right\rfloor = \mathbf{H} * \lceil \mathbf{x}_1^{n_1}, \mathbf{x}_2^{n_2}, \cdots, \mathbf{x}_m^{n_m} \rfloor.$$

In the following, we will illustrate two observations of $\mathbf{H} * \mathbf{x}^n$, where $\mathbf{x}$ is *discrete one-dimensional* signal.

The first observation is that each dimension of $\mathbf{H}$ is equal, i.e., $\mathbf{H} \in \mathbb{R}^{m \times m \times \cdots \times m}$. The second observation is that there exists symmetric $\widehat{\mathbf{H}}$ such that $\widehat{\mathbf{H}} * \mathbf{x}^n = \mathbf{H} * \mathbf{x}^n$, where symmetry means $\hat{H}(\cdots, \tau_i, \cdots, \tau_j, \cdots) = \hat{H}(\cdots, \tau_j, \cdots, \tau_i, \cdots)$ for any $\tau_i, \tau_j$ at any dimension. The first is obvious, and the second is demonstrated as below.

Expanding $\mathbf{H} * \mathbf{x}^n$ as

$$\begin{aligned} &\cdots + H(\cdots, \tau_i, \cdots, \tau_j, \cdots) \cdots x(t - \tau_i) \cdots x(t - \tau_j) \cdots \\ &\quad + H(\cdots, \tau_j, \cdots, \tau_i, \cdots) \cdots x(t - \tau_j) \cdots x(t - \tau_i) \cdots + \cdots, \end{aligned}$$

and take $\hat{H}(\cdots,\tau_i,\cdots,\tau_j,\cdots) = \frac{1}{2}\left(H(\cdots,\tau_i,\cdots,\tau_j,\cdots)+H(\cdots,\tau_i,\cdots,\tau_j,\cdots)\right)$, we have $\widehat{\mathbf{H}} * \mathbf{x}^n = \mathbf{H} * \mathbf{x}^n$. Therefore, without further notice, this kind of kernels are always symmetric.

### 2.2 Volterra Convolution

In this subsection, we sum these convolutions from order-zero to order-$n$ or order-$\infty$. If the order is finite, it is called the finite term Volterra convolution or order-$n$ Volterra convolution, otherwise it is called the infinity term Volterra convolution or Volterra convolution. For instance, the order-two Volterra convolution is sum of order-zero, order-one and order-two convolutions, $\mathbf{H}_0 * \mathbf{x}^0 + \mathbf{H}_1 * \mathbf{x}^1 + \mathbf{H}_2 * \mathbf{x}^2$.

For simplicity and the fact that a neural network takes only one input (multiple inputs are packed to one tensor), input signals of each order are set to be the same $\mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_n$. If the input signals are one-dimensional, all kernels are symmetric.

**Definition 3** *Let* $\mathbf{x}$ *be signal and* $\mathbf{H}_n$ *as kernels, Volterra convolution (Volterra, 1932; Rugh, 1981) is defined as*

$$\sum_{n=0}^{+\infty} \mathbf{H}_n * \mathbf{x}^n = \sum_{n=0}^{+\infty} \mathbf{H}_n * \underbrace{\lceil \mathbf{x}, \mathbf{x}, \cdots \rfloor}_{n \text{ terms}}. \tag{8}$$

*If* $n = 0$, $\mathbf{x}^0 = \delta$, *i.e., the Dirac delta.*

If $\mathbf{x}$ is a one-dimensional signal and each $\mathbf{H}_n$ is an $n$-dimensional signal, continuous Volterra convolution for one-dimensional signal is

$$\left(\sum_{n=0}^{+\infty} \mathbf{H}_n * \mathbf{x}^n\right)(t) = \sum_{n=0}^{+\infty} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} H_n(\tau_1,\cdots,\tau_n) \prod_{i=1}^{n} \left(x(t-\tau_i)d\tau_i\right). \tag{9}$$

According to previous discussion, here the kernels $\mathbf{H}_n$, $n = 1, 2, 3, \ldots$, are symmetric.

If $\mathbf{x}$ is an $m$-dimensional signal and $\mathbf{H}_n$ is an $nm$-dimensional signal for each $n = 0, 1, \cdots$, and $\vec{\tau}_1 = \lceil \tau_{1,1}, \tau_{1,2}, \cdots, \tau_{1,m} \rfloor; \cdots; \vec{\tau}_n = \lceil \tau_{n,1}, \tau_{n,2}, \cdots, \tau_{n,m} \rfloor$ and $\vec{\mathbf{t}} = \lceil t_1, t_2, \cdots, t_m \rfloor$, continuous Volterra convolution for $m$-dimensional signal is

$$\begin{aligned}\left(\sum_{n=0}^{+\infty} \mathbf{H}_n * \mathbf{x}^n\right)(t_1,\cdots,t_m) = &\sum_{n=0}^{+\infty} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} H_n(\tau_{1,1},\cdots,\tau_{1,m};\cdots;\tau_{n,1},\cdots,\tau_{n,m}) \\ &\prod_{i=1}^{n} \left(x(t_1-\tau_{i,1},\cdots,t_m-\tau_{i,m})d\tau_{i,1}\cdots d\tau_{i,m}\right) \\ \equiv &\sum_{n=0}^{+\infty} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} H_n(\vec{\tau}_1,\vec{\tau}_2,\cdots,\vec{\tau}_n) \prod_{i=1}^{n} \left(x(\vec{\mathbf{t}}-\vec{\tau}_i)d\vec{\tau}_i\right).\end{aligned} \tag{10}$$

With the vector like notation, discrete Volterra convolution for $m$-dimensional signal is simplified as

$$\left(\sum_{n=0}^{+\infty} \mathbf{H}_n * \mathbf{x}^n\right)(\vec{\mathbf{t}}) = \sum_{n=0}^{+\infty} \sum_{\vec{\tau}_1,\cdots,\vec{\tau}_n} H_n(\vec{\tau}_1,\vec{\tau}_2,\cdots,\vec{\tau}_n) \prod_{i=1}^{n} x(\vec{\mathbf{t}}-\vec{\tau}_i). \tag{11}$$

### 2.3 Outer Convolution

Stacking two order-one one-dimensional convolutions will produce a one-dimensional convolution with a longer kernel. How do we stack order-$n$ convolutions? In this subsection, an operation, the outer convolution, is introduced to combine these convolutions. Moreover, the rank properties for outer convolutions are described in Appendix D.

Let $\mathbf{G}$ and $\lceil \mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_n \rfloor \equiv \vec{\mathbf{H}}$ are the kernels for convolutions of one-dimensional signal. The outer convolution of $\mathbf{G}$ and $\vec{\mathbf{H}}$, denoted by

$$\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_n \rfloor \equiv \mathbf{G} \circledast \vec{\mathbf{H}}, \tag{12}$$

is defined as follows.

**Definition 4 (Continuous outer convolution of kernels)** *Let $\mathbf{G}$ be an $n$-dimensional kernel and each dimension of $\mathbf{H}_i$ no less than one, then the continuous outer convolution yields an $L$-dimensional kernel satisfying*

$$\begin{aligned}
&\left(\mathbf{G} \circledast \vec{\mathbf{H}}\right)(t_{1,1}, t_{1,2}, \cdots; t_{2,1}, t_{2,2}, \cdots; \cdots; t_{n,1}, t_{n,2}, \cdots) \\
&= \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} G(\tau_1, \tau_2, \cdots, \tau_n) \prod_{i=1}^{n} (H_i(t_{i,1} - \tau_i, t_{i,2} - \tau_i, \cdots) d\tau_i) \\
&\equiv \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} G(\vec{\tau}) \prod_{i=1}^{n} \left(H_i(\vec{\mathbf{t}}_i - \tau_i) d\tau_i\right),
\end{aligned} \tag{13}$$

*where $L$ is the sum of the dimensions of $\mathbf{H}_i$, $\vec{\tau} = \lceil \tau_1, \tau_2, \cdots, \tau_n \rfloor$, $\vec{\mathbf{t}}_1 = \lceil t_{1,1}, t_{1,2}, \cdots \rfloor$, $\vec{\mathbf{t}}_2 = \lceil t_{2,1}, t_{2,2}, \cdots \rfloor$, $\cdots$, $\vec{\mathbf{t}}_n = \lceil t_{n,1}, t_{n,2}, \cdots \rfloor$, using the vector like notations.*

With Equation 13, we can compute the convolution between $\mathbf{G} \circledast \vec{\mathbf{H}}$ and one-dimensional signals $\mathbf{x}$. Since most signals in this article are one-dimensional, we prefer to express the outer convolution in the form of Equation 13.

Note that Equation 13 allows us to combine multiple convolution layers, and detailed rules will be described in Subsection 2.7. Two of these rules are quick previewed as follows:

- $\mathbf{G} * (\mathbf{H} * \mathbf{x}) = (\mathbf{G} \circledast \mathbf{H}) * \mathbf{x}$ (Property 7),
- $\mathbf{G} * \lceil \mathbf{H}_1 * \mathbf{x}_1, \mathbf{H}_2 * \mathbf{x}_2 \rfloor = (\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor) * \lceil \mathbf{x}_1, \mathbf{x}_2 \rfloor$ (Property 9).

More generally, we consider the layers involving the convolution of $m$-dimensional signals $\mathbf{x}$. To this end, suppose that $\mathbf{G}$ is an $nm$-dimensional kernel, and each dimension of $\mathbf{H}_i$ is

no less than $m$, continuous outer convolution of $\mathbf{G}$ and $\mathbf{H}_i$ can be expressed as

$$\begin{aligned}
&\left(\mathbf{G} \circledast \vec{\mathbf{H}}\right) \begin{pmatrix} t_{1,1,1}, t_{1,1,2}, \cdots t_{1,1,m}; t_{1,2,1}, \cdots t_{1,2,m}; \cdots ; \\ t_{2,1,1}, t_{2,1,2}, \cdots t_{2,1,m}; t_{2,2,1}, \cdots t_{2,2,m}; \cdots ; \\ \cdots ; \cdots ; \\ t_{n,1,1}, t_{n,1,2}, \cdots t_{n,1,m}; t_{n,2,1}, \cdots t_{n,2,m}; \cdots ; \end{pmatrix} \\
&= \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} G\left(\tau_{1,1}, \tau_{1,2}, \cdots, \tau_{1,m}; \tau_{2,1}, \tau_{2,2}, \cdots, \tau_{2,m}; \cdots ; \cdots ; \tau_{n,1}, \tau_{n,2}, \cdots, \tau_{n,m};\right) \\
&\quad \prod_{i=1}^{n} \left( H_i \begin{pmatrix} t_{i,1,1} - \tau_{i,1}, t_{i,1,2} - \tau_{i,2}, \cdots, t_{i,1,m} - \tau_{i,m}; \\ t_{i,2,1} - \tau_{i,1}, t_{i,2,2} - \tau_{i,2}, \cdots, t_{i,2,m} - \tau_{i,m}; \\ \cdots \end{pmatrix} d\tau_{i,1} d\tau_{i,2} \cdots d\tau_{i,m} \right). \\
&\equiv \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} G(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} \left( H_i(\vec{\mathbf{t}}_{i,1} - \vec{\tau}_i, \vec{\mathbf{t}}_{i,2} - \vec{\tau}_i, \cdots) d\vec{\tau}_i \right).
\end{aligned} \tag{14}$$

The outer convolution (Equation 14) will yield a new kernel whose convolution with $m$-dimentional signals is given by Equation 6.

Using the vector like notation, discrete outer convolution for one-dimensional signal is simplified as

$$\left(\mathbf{G} \circledast \vec{\mathbf{H}}\right) (\vec{\mathbf{t}}_1, \vec{\mathbf{t}}_2, \cdots, \vec{\mathbf{t}}_n) = \sum_{\vec{\tau}} G(\vec{\tau}) \prod_{i=1}^{n} H_i(\vec{\mathbf{t}}_i - \tau_i), \tag{15}$$

and discrete outer convolution for $m$-dimensional signal is simplified as

$$\begin{aligned}
&\left(\mathbf{G} \circledast \vec{\mathbf{H}}\right) (\vec{\mathbf{t}}_{1,1}, \vec{\mathbf{t}}_{1,2}, \cdots ; \vec{\mathbf{t}}_{2,1}, \cdots ; \cdots ; \vec{\mathbf{t}}_{n,1}, \cdots) \\
&= \sum_{\vec{\tau}_1, \cdots} G(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} H_i(\vec{\mathbf{t}}_{i,1} - \vec{\tau}_i, \vec{\mathbf{t}}_{i,2} - \vec{\tau}_i, \cdots).
\end{aligned} \tag{16}$$

The shorthand notations of outer convolution are similar to that of order-$n$ convolution. If all signals are equal $\mathbf{H}_1 = \mathbf{H}_2 = \cdots = \mathbf{H}_n$, we have $\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_n \rfloor = \mathbf{G} \circledast \mathbf{H}^n$.

If signals are grouped and $n_1 + n_2 + \cdots + n_m$ equals the order, the notation can be simplified as

$$\mathbf{G} \circledast \left[ \underbrace{\mathbf{H}_1, \cdots, \mathbf{H}_1}_{n_1 \text{ terms}}, \underbrace{\mathbf{H}_2, \cdots, \mathbf{H}_2}_{n_2 \text{ terms}}, \cdots, \underbrace{\mathbf{H}_m, \cdots, \mathbf{H}_m}_{n_m \text{ terms}} \right] = \mathbf{G} * \lceil \mathbf{H}_1^{n_1}, \mathbf{H}_2^{n_2}, \cdots, \mathbf{H}_m^{n_m} \rfloor .$$

**Remark 5** *In discrete outer convolution $\mathbf{G} \circledast \vec{\mathbf{H}}$, kernels $\mathbf{H}_1, \cdots, \mathbf{H}_n$ are all zero padded on both heads and tails. The padding size of $\mathbf{H}_i$ is $s_i - 1$, where $s_i$ is shape of $\mathbf{G}$ at dimension $i$. For instance, one-dimensional $\mathbf{H}_i$ is zero padded as*

$$\left[ \underbrace{\cdots\ 0\ \cdots}_{s_i - 1} \quad H_i(0) \cdots H_i(n) \quad \underbrace{\cdots\ 0\ \cdots}_{s_i - 1} \right].$$

For example, if $\mathbf{G}$ has shape $(s_1, s_2)$, $\mathbf{H}_1$ has shape $(z_1)$ and $\mathbf{H}_2$ has shape $(z_2, z_3, z_4)$, outer convolution (with zero padded) $\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor$ will have shape $(s_1 + z_1 - 1, s_2 + z_2 - 1, s_2 + z_3 - 1, s_2 + z_4 - 1)$.

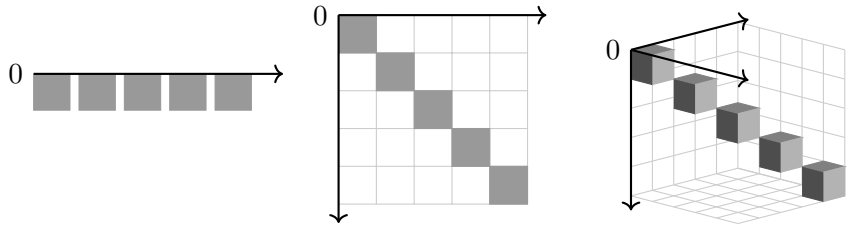
**Remark 6** $\mathbf{G} \circledast \mathbf{H}$ *has the same operation as the "ConvTranspose", which is a deep learning operator* [1].

**Remark 7** *If* $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n$ *are all one-dimensional vectors, and tensor* $H(t_1, t_2, \cdots, t_n) = h_1(t_1)h_2(t_2)\cdots h_n(t_n)$*, outer convolution can be transformed to multidimensional convolution* $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n \rfloor = \mathbf{G} * \mathbf{H}$.

The super diagonal kernel $\mathbf{G}$ has only non-zero elements on the diagonal, $G(0, 0, \cdots, 0)$, $G(1, 1, \cdots, 1)$, $\cdots$, and all other elements are set to zero. More specifically, we define the $\mathbf{G} = \mathrm{diag}(n, \mathbf{g})$, where $\mathbf{g}$ is a vector, and

$$G(\tau_1, \tau_2, \cdots, \tau_n) = \mathrm{diag}(n, \mathbf{g})(\tau_1, \tau_2, \cdots, \tau_n) = \sum_k g(k) \prod_{i=1}^{n} \delta(\tau_i - k). \tag{17}$$

For better understanding, this diag$(\cdot)$ operator is visualized in Figure 2.




(a) one-dimensional $\mathbf{g} \in \mathbb{R}^5$ (b) $\mathrm{diag}(2, \mathbf{g}) \in \mathbb{R}^{5\times5}$ (c) $\mathrm{diag}(3, \mathbf{g}) \in \mathbb{R}^{5\times5\times5}$

Figure 2: A brief preview of the diag$(\cdot)$ operator.

Specially, if $\mathbf{G}$ is a super diagonal tensor kernel, its outer convolution has the form of

$$\begin{aligned}
&\left(\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_n \rfloor\right)(\vec{\mathbf{t}}_1, \vec{\mathbf{t}}_2, \cdots, \vec{\mathbf{t}}_n) \\
&= \sum_{\vec{\tau}} G(\vec{\tau}) \prod_{i=1}^{n} H_i(\vec{\mathbf{t}}_i - \tau_i) = \sum_k g(k) \prod_{i=1}^{n} \delta(\tau_i - k) H_i(\vec{\mathbf{t}}_i - \tau_i) = \sum_k g(k) \prod_{i=1}^{n} H_i(\vec{\mathbf{t}}_i - k).
\end{aligned} \tag{18}$$

### 2.4 Convolution With Stride Grater Than One

Convolution with stride grater than one is commonly used to replace convolution-pooling structure. If the stride equals one, filters move one point at a time. If the stride equals two, filters jump two points at a time. In addition, convolution with stride $s$ is

$$\left(\mathbf{h} *_s \mathbf{x}\right)(t) = \sum_{\tau} h(\tau) x(st - \tau),$$

where subscript of asterisk $*_s$ indicates stride.

1. `https://pytorch.org/docs/stable/generated/torch.nn.functional.conv_transpose1d.html`

This operation can be applied to order-$n$ convolution and outer convolution. With vector like notation, let $s\vec{\mathbf{t}} = \lceil st_1, st_2, \cdots, st_n \rfloor$. Discrete order-$n$ convolution for $m$-dimensional signal with stride $s$ is

$$\left(\mathbf{H} *_s \vec{\mathbf{x}}\right)\left(\vec{\mathbf{t}}\right) = \sum_{\vec{\tau}_1, \cdots, \vec{\tau}_n} H(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} x_i(s\vec{\mathbf{t}} - \vec{\tau}_i). \tag{19}$$

Similarly, the discrete outer convolution for $m$-dimensional signal with stride $s$ is

$$\begin{aligned} &\left(\mathbf{G} \circledast_s \vec{\mathbf{H}}\right)\left(\vec{\mathbf{t}}_{1,1}, \vec{\mathbf{t}}_{1,2}, \cdots; \vec{\mathbf{t}}_{2,1}, \cdots; \cdots; \vec{\mathbf{t}}_{n,1}, \cdots\right) \\ &= \sum_{\vec{\tau}_1, \cdots} G(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} H_i(s\vec{\mathbf{t}}_{i,1} - \vec{\tau}_i, s\vec{\mathbf{t}}_{i,2} - \vec{\tau}_i, \cdots). \end{aligned} \tag{20}$$

The combination of two convolutions with strides is also equivalent to the outer convolution with strides, $\mathbf{G} *_s (\mathbf{H} *_z \mathbf{x}) = (\mathbf{G} \circledast_z \mathbf{H}) *_{sz} \mathbf{x}$ (Property 8 in Subsection 2.7).

**Remark 8** *If* $\mathbf{G}$ *has shape* $(z_1, z_2, \cdots, z_m)$ *and* $\mathbf{H}$ *has shape* $c_1, c_2, \cdots, c_m$*, the shape of* $\mathbf{G} \circledast_s \mathbf{H}$ *is*

$$\left(c_1 + (z_1 - 1)s, c_2 + (z_2 - 1)s, \cdots, c_m + (z_m - 1)s\right).$$

### 2.5 Visualization of Outer Convolution

For the following cases, three examples are provided to help understand the outer convolution.

Let $\mathbf{G} \in \{1\}^{8\times 8}$ be a $8 \times 8$ matrix with all elements are one, and $\mathbf{h} \in \{1\}^8$ be a vector of length eight with all elements are one. Figure 3a demonstrates

$$\left(\mathbf{G} \circledast \lceil \mathbf{h}, \mathbf{h} \rfloor\right)(t_1, t_2) = \sum_{\tau_1=0, \tau_2=0}^{7,7} G(\tau_1, \tau_2) h(t_1 - \tau_1) h(t_2 - \tau_2).$$

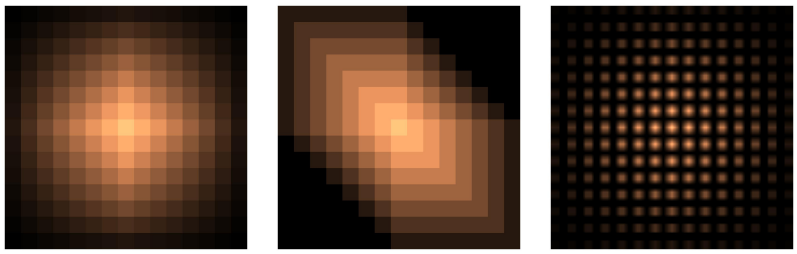
Let $\mathbf{g} \in \{1\}^8$ and $\mathbf{H} \in \{1\}^{8\times 8}$. Figure 3b demonstrates

$$\left(\mathbf{g} \circledast \mathbf{H}\right)(t_1, t_2) = \sum_{t_1=0, t_2=0}^{7,7} g(\tau) H(t_1 - \tau, t_2 - \tau).$$

Assume $\mathbf{G} \in \{1\}^{8\times 8}$ and $\mathbf{H} \in \{1\}^{8\times 8}$, Figure 3c demonstrates

$$\left(\mathbf{G} \circledast \lceil \mathbf{H}, \mathbf{H} \rfloor\right)(t_1, t_2, t_3, t_4) = \sum_{\tau_1=0, \tau_2=0}^{7,7} G(\tau_1, \tau_2) H(t_1 - \tau_1, t_2 - \tau_1) H(t_3 - \tau_2, t_4 - \tau_2).$$

It is clear that $\mathbf{G} \circledast \lceil \mathbf{H}, \mathbf{H} \rfloor$ is a four-dimensional tensor with shape $(15, 15, 15, 15)$, and it is flattened to shape $(225, 225)$ and drawn in Figure 3c.



(a) $\mathbf{G} \circledast \lceil \mathbf{h}, \mathbf{h} \rfloor$ (b) $\mathbf{g} \circledast \mathbf{H}$ (c) flattened $\mathbf{G} \circledast \lceil \mathbf{H}, \mathbf{H} \rfloor$

Figure 3: Examples of outer Convolution.

### 2.6 Convolution for Multi-dimensional Signals

In this subsection, we show that the multidimensional (outer) convolution can be analyzed via one-dimensional (outer) convolution. With this transformation, only (outer) convolution for one-dimensional signals will be studied hereafter, if not specified.

**Definition 9 (Flatten-operator)** *The flatten-operator is a bijection $\mathcal{T}(\mathbf{x}) = \widehat{\mathbf{x}}$, such that $x(t_1, t_2, \cdots) = \hat{x}(t_1 w_1 + t_2 w_2 + \cdots)$ and $w_1, w_2$ are scalars for locating non-overlap elements. The inverse of $\mathcal{T}$ is denoted by $\mathcal{T}^{-1}$ with $\mathcal{T}^{-1}(\mathcal{T}(\mathbf{x})) = \mathbf{x}$.*

**Proposition 10** *The flatten-operator is homomorphic,*

$$\begin{aligned} \mathcal{T}(\mathbf{H} * \vec{\mathbf{x}}) &= \mathcal{T}(\mathbf{H}) * \mathcal{T}(\vec{\mathbf{x}}), \\ \mathcal{T}(\mathbf{G} \circledast \vec{\mathbf{H}}) &= \mathcal{T}(\mathbf{G}) \circledast \mathcal{T}(\vec{\mathbf{H}}). \end{aligned} \tag{21}$$

**Proof** A continuous case of this proposition is proved here, and the discrete one can be obtained in the similar way. Recall order-$n$ convolution for $m$-dimensional signal (Equation 7), and let flattened index $\vec{\tau}_i$ be $\upsilon_i$, flattened index $\vec{\mathbf{t}}$ be $\iota$. We have

$$\begin{aligned} \mathcal{T}(\mathbf{H} * \vec{\mathbf{x}})(\vec{\mathbf{t}}) &= \mathcal{T}\left( \sum_{\vec{\tau}_i, \cdots, \vec{\tau}_n} H(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} x_i(\vec{\mathbf{t}} - \vec{\tau}_i) \right) \\ &= \sum_{\upsilon_1, \cdots, \upsilon_n} \hat{H}(\upsilon_1, \upsilon_2, \cdots, \upsilon_n) \prod_{i=1}^{n} \hat{x}_i(\iota - \upsilon_i) \\ &= \left(\mathcal{T}(\mathbf{H}) * \mathcal{T}(\vec{\mathbf{x}})\right)(\iota). \end{aligned}$$

The flatten-operator of outer convolution for $m$-dimensional signal is also homomorphic. Recall outer convolution for $m$-dimensional signal (Equation 11), and let flattened index $\vec{\mathbf{t}}_{i,j}$

be $\iota_{i,j}$, flattened index $\vec{\tau}_i$ be $\upsilon_i$. We have

$$\begin{aligned}
&\mathcal{T}(\mathbf{G} \circledast \vec{\mathbf{H}})(\vec{\mathbf{t}}_{1,1}, \vec{\mathbf{t}}_{1,2}, \cdots; , \vec{\mathbf{t}}_{2,1}, \cdots; \cdots; \vec{\mathbf{t}}_{n,1}, \cdots) \\
&= \sum_{\vec{\tau}_1, \cdots, \vec{\tau}_n} G(\vec{\tau}_1, \vec{\tau}_2, \cdots, \vec{\tau}_n) \prod_{i=1}^{n} H_i(\vec{\mathbf{t}}_{i,1} - \vec{\tau}_i, \vec{\mathbf{t}}_{i,2} - \vec{\tau}_i, \cdots) \\
&= \sum_{\upsilon_1, \cdots, \upsilon_n} \hat{G}(\upsilon_1, \upsilon_2, \cdots, \upsilon_n) \prod_{i=1}^{n} \hat{H}_i(\iota_{i,1} - \upsilon_1, \iota_{i,2} - \upsilon_1, \cdots) \\
&= \left(\mathcal{T}(\mathbf{G}) \circledast \mathcal{T}(\vec{\mathbf{H}})\right)(\iota_{1,1}, \iota_{1,2}, \cdots; \iota_{2,1}, \cdots; \cdots; \iota_{n,1}, \cdots).
\end{aligned}$$

■

With Proposition 10, it could be easily verified

$$\begin{aligned}
\mathbf{H} * \vec{\mathbf{x}} &= \mathcal{T}^{-1}\left(\mathcal{T}(\mathbf{H}) * \mathcal{T}(\vec{\mathbf{x}})\right), \\
\mathbf{G} \circledast \vec{\mathbf{H}} &= \mathcal{T}^{-1}\left(\mathcal{T}(\mathbf{G}) \circledast \mathcal{T}(\vec{\mathbf{H}})\right).
\end{aligned}$$

To help visualize this process, Figure 4 is an example of flattening a two-dimensional signal to a one-dimensional signal.

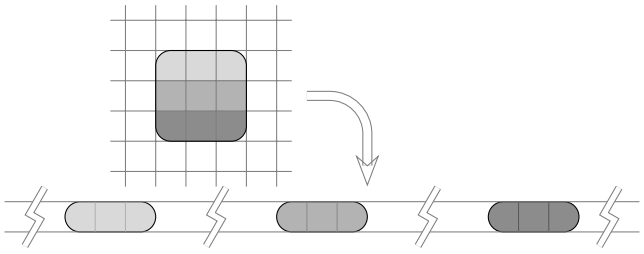


Figure 4: Flattening a two-dimensional signal to a one-dimensional signal.

### 2.7 Combination Properties

With all definitions above, some useful properties are concluded in this subsection. All proofs are presented in Appendix B.

Since the multidimensional signals can be analyzed via one-dimensional signals, all signals here are set to be *one-dimensional* and are notated as $\mathbf{x}$ or $\mathbf{y}$. $\mathbf{H}$ and $\mathbf{G}$ are kernels. $\alpha$ is a scalar.

The following Property 1, 2 and 3 are about linearity of order-$n$ convolutions. In continuous time, notation $\sum \mathbf{G}$ is replaced with $\int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} G(\vec{\mathbf{t}}) d\vec{\mathbf{t}}$.

1. $\mathbf{G} * (\vec{\mathbf{x}} + \vec{\mathbf{y}}) = \mathbf{G} * \vec{\mathbf{x}} + \mathbf{G} * \vec{\mathbf{y}}$;
2. $\mathbf{G} * (\vec{\mathbf{x}} + \alpha) = \mathbf{G} * \vec{\mathbf{x}} + \alpha \sum \mathbf{G}$;
3. $(\mathbf{G} + \mathbf{H}) * \vec{\mathbf{x}} = \mathbf{G} * \vec{\mathbf{x}} + \mathbf{H} * \vec{\mathbf{x}}$;

Property 4, 5 and 6 are combination properties of order-$n$ convolution. Based on previous discussion, the kernel $\mathbf{G}$ here is symmetric. The multinomial coefficient in Property 6 can be obtained from textbook of combinatorial mathematic (Brualdi, 2004).

4. $\mathbf{G} * (\mathbf{x} + \mathbf{y})^2 = \mathbf{G} * \mathbf{x}^2 + 2\mathbf{G} * \lceil \mathbf{x}, \mathbf{y} \rfloor + \mathbf{G} * \mathbf{y}^2$;
5. $\mathbf{G} * (\mathbf{x}+\mathbf{y})^n = \sum_{k=0}^{n} \binom{n}{k} \mathbf{G} * \left\lceil \mathbf{x}^k, \mathbf{y}^{n-k} \right\rfloor$, where $\binom{n}{k} = \dfrac{n!}{k!(n-k)!}$ is binomial coefficient;
6. $\mathbf{G} * (\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_m)^n = \sum \binom{n}{n_1 n_2 \cdots n_m} \mathbf{G} * \lceil \mathbf{x}_1^{n_1}, \mathbf{x}_2^{n_2}, \cdots, \mathbf{x}_m^{n_m} \rfloor$, where multinomial coefficient $\binom{n}{n_1 n_2 \cdots n_m} = \dfrac{n!}{n_1! n_2! \cdots n_m!}$, and $\sum_{i=1}^{m} n_i = m, n_i \geq 0$, for all $i = 1, 2, \cdots, m$.

Properties below are for combining convolutions. The symbol "#" indicates summation along a specific dimension. For example, $\sum_{\#\alpha} (\mathbf{G} \circledast \lceil \mathbf{H}, \alpha \rfloor) = \sum_i (\mathbf{G} \circledast \lceil \mathbf{H}, \alpha \rfloor)(:, i)$ and $\sum_{\#\alpha} (\mathbf{G} \circledast \lceil \mathbf{H}_1, \alpha, \mathbf{H}_2 \rfloor) = \sum_i (\mathbf{G} \circledast \lceil \mathbf{H}_1, \alpha, \mathbf{H}_2 \rfloor)(:, i, :)$. In continuous space, they are replaced by $\int_{-\infty}^{+\infty} (\mathbf{G} \circledast \lceil \mathbf{H}, \alpha \rfloor)(:, t_\alpha) dt_\alpha$ and $\int_{-\infty}^{+\infty} (\mathbf{G} \circledast \lceil \mathbf{H}_1, \alpha, \mathbf{H}_2 \rfloor)(:, t_\alpha, :) dt_\alpha$.

7. $\mathbf{G} * (\mathbf{H} * \vec{\mathbf{x}}) = (\mathbf{G} \circledast \mathbf{H}) * \vec{\mathbf{x}}$;
8. $\mathbf{G} *_s (\mathbf{H} *_z \mathbf{x}) = (\mathbf{G} \circledast_z \mathbf{H}) *_{sz} \mathbf{x}$;
9. $\mathbf{G} * \lceil \mathbf{H}_1 * \vec{\mathbf{x}}, \mathbf{H}_2 * \vec{\mathbf{y}} \rfloor = (\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor) * \lceil \vec{\mathbf{x}}, \vec{\mathbf{y}} \rfloor$;
10. $\mathbf{G} * \lceil \mathbf{H}_1 * \vec{\mathbf{x}}_1, \mathbf{H}_2 * \vec{\mathbf{x}}_2, \cdots \rfloor = (\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2, \cdots \rfloor) * \lceil \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \cdots \rfloor$;
11. $\mathbf{G}_1 \circledast (\mathbf{G}_2 \circledast \mathbf{G}_3) = (\mathbf{G}_1 \circledast \mathbf{G}_2) \circledast \mathbf{G}_3$;
12. $\mathbf{G} * \lceil \alpha, \mathbf{H} * \vec{\mathbf{x}} \rfloor = \left( \sum_{\#\alpha} (\mathbf{G} \circledast \lceil \alpha, \mathbf{H} \rfloor) \right) * \vec{\mathbf{x}}$;
13. $\mathbf{G} * \lceil \mathbf{H} * \vec{\mathbf{x}}, \alpha \rfloor = \left( \sum_{\#\alpha} (\mathbf{G} \circledast \lceil \mathbf{H}, \alpha \rfloor) \right) * \vec{\mathbf{x}}$;
14. $\mathbf{G} * \lceil \mathbf{H}_1 * \vec{\mathbf{x}}, \alpha, \mathbf{H}_2 * \vec{\mathbf{y}} \rfloor = \left( \sum_{\#\alpha} (\mathbf{G} \circledast \lceil \mathbf{H}_1, \alpha, \mathbf{H}_2 \rfloor) \right) * \lceil \vec{\mathbf{x}}, \vec{\mathbf{y}} \rfloor$;

Property 15 and 16 focus on convolution with signal that is element-wise power $n$, $([\mathbf{x}]^n)(t) = (x(t))^n$.

15. $\mathbf{h} * [\mathbf{x}]^n = \mathrm{diag}(n, \mathbf{h}) * \mathbf{x}^n$;
16. $\mathbf{g} * [\mathbf{h} * \mathbf{x}]^n = (\mathrm{diag}(n, \mathbf{g}) \circledast \mathbf{h}^n) * \mathbf{x}^n$.

In addition to all properties above, some special properties could be obtained via setting special kernels. For example, by setting $\mathbf{G}$ as the identity matrix, we have

$$(\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor)(\vec{\mathbf{t}}_1, \vec{\mathbf{t}}_2) = \sum_{i_1, i_2} G(i_1, i_2) H_1(\vec{\mathbf{t}}_1 - i_1) H_2(\vec{\mathbf{t}}_2 - i_2) = H_1(\vec{\mathbf{t}}_1) H_2(\vec{\mathbf{t}}_2).$$

## 3. Transformation from Neural Networks to Volterra Convolutions

Previous section discussed the definition of Volterra convolution and some useful properties of combining two order-$n$ convolutions. In this section, we will go further and try to represent some common convolutional networks in the form of Volterra convolutions.

**Theorem 11** *Most convolutional neural networks can be represented in the form of Volterra convolutions.*

Both convolutional networks and Volterra convolutions have the operation of convolution. The convolutional neural network extend this operation by stacking layers and the Volterra convolution extend this by increasing the order. Apart from the convolution, a convolutional neural network is a universal approximator, as it happens, a Volterra convolution is also a universal approximator. Theoretically, if two approximators can approximate the same function, it is possible to approximate one by the other. In light of this, roughly speaking, most convolutional neural network can be approximated in the form of Volterra convolution, and vice versa.

The proof contains two major parts. The first part is about the small neural network structures, and the second part is about the combination of multiple layers, i.e., the whole network. Since both the small structures and their combinations can be represented in this form, we conclude that most convolutional neural networks build of these structures can also be represented in the form of Volterra convolution.

### 3.1 Conversion of Small Structures

#### 3.1.1 Conv-Act-Conv Structure

The "conv — act — conv" structure means the stacking of a convolutional layer, an activation layer, and a convolutional layer.

**Lemma 12** *The "conv — act — conv" structure can be converted to the form of Volterra convolution.*

**Proof** Suppose this structure has the form $\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})$, where $\sigma(\cdot)$ is a nonlinear activation function.

A polynomial approximation, i.e., Taylor expansion, of this activation function $\sigma(t)$ at $\alpha$ is

$$\sigma(t) = \sigma(\alpha) + \sigma'(\alpha)(t-\alpha) + \frac{\sigma''(\alpha)}{2!}(t-\alpha)^2 + \frac{\sigma'''(\alpha)}{3!}(t-\alpha)^3 + \cdots .$$

We can assume without loss of generality that $\alpha = 0$, and $\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})$ becomes

$$\mathbf{g} * \left( \sigma(0) + \sigma'(0)[\mathbf{h} * \mathbf{x}] + \frac{\sigma''(0)}{2!}[\mathbf{h} * \mathbf{x}]^2 + \frac{\sigma'''(0)}{3!}[\mathbf{h} * \mathbf{x}]^3 + \cdots \right), \tag{22}$$

where the square brackets stand for

$$[\mathbf{h} * \mathbf{x}]^n(t) = \left( \sum_{\tau} h(\tau) x(t-\tau) \right)^n .$$

By linearity of convolution, Equation 22 can be separated by terms. The first term is $\sum \mathbf{g}$. The second term is $\mathbf{g} * [\mathbf{h} * \mathbf{x}]^1 = (\mathbf{g} \circledast \mathbf{h}) * \mathbf{x}$. For the third term and above, recall Property 16, we have the $n$-th term,

$$\mathbf{g} * [\mathbf{h} * \mathbf{x}]^n = (\mathrm{diag}(n, \mathbf{g}) \circledast \mathbf{h}^n) * \mathbf{x}^n.$$

If $\alpha = 0$, we conclude that

$$\begin{aligned} \mathbf{g} * \sigma(\mathbf{h} * \mathbf{x}) = \sigma(0) \sum \mathbf{g} + \sigma'(0)(\mathbf{g} \circledast \mathbf{h}) * \mathbf{x} \\ + \frac{\sigma''(0)}{2!}(\mathrm{diag}(2, \mathbf{g}) \circledast \mathbf{h}^2) * \mathbf{x}^2 + \frac{\sigma'''(0)}{3!}(\mathrm{diag}(3, \mathbf{g}) \circledast \mathbf{h}^3) * \mathbf{x}^3 + \cdots . \end{aligned} \tag{23}$$

More generally, if $\alpha \neq 0$, the $n$-th term is

$$\begin{aligned} &\mathbf{g} * [\mathbf{h} * \mathbf{x} - \alpha]^n \\ &= \mathrm{diag}(n, \mathbf{g}) * (\mathbf{h} * \mathbf{x} - \alpha)^n \quad \text{(Property 15)} \\ &= \sum_{k=0}^{n} \binom{n}{k} \mathrm{diag}(n, \mathbf{g}) * \left\lceil (\mathbf{h} * \mathbf{x})^k, (-\alpha)^{n-k} \right\rfloor \quad \text{(Property 5)} \\ &= \sum_{k=0}^{n} \binom{n}{k} \left( \sum_{\#(-\alpha)^{n-k}} \left( \mathrm{diag}(n, \mathbf{g}) \circledast \left\lceil \mathbf{h}^k, (-\alpha)^{n-k} \right\rfloor \right) \right) * \mathbf{x}^k \quad \text{(Property 13)}, \end{aligned}$$

which implies that the sum from the 0-th term to the $\infty$-th term is also the form of Volterra convolution. This proof is completed. ■

The Taylor expansion of a function often has infinite terms. However, if small truncation errors are allowed in applications, we can truncate the infinite term Taylor expansion to a finite term Taylor expansion. The idea of truncation can also be applied to the Volterra convolution. According to the universal approximation property of Volterra convolution with fading memory (Boyd and Chua, 1985), for any given $\epsilon > 0$, there always exists $n$ such that

$$\left\| f(\mathbf{x}) - \sum_{i=0}^{n} \mathbf{F}_i * \mathbf{x}^i \right\|_2 < \epsilon,$$

where $f(\cdot)$ is a time invariant operation, and $\mathbf{F}_i, i = 0, 1, \cdots n$ are kernels. The fading memory theory means that the outputs are close if two inputs are close in the recent past, but not necessarily close in the remote past (Boyd and Chua, 1985). If $\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})$ is time invariant and small truncation errors are allowed, it is reasonable to approximate $\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})$ by a finite term Volterra convolution.

In the following, we will consider the width of a two-layer network. Suppose a two-layer network has $M$ hidden neurons, and the neurons of distinct channels are independent and identically distributed,

$$y(t) = \sum_{i=0}^{M-1} W_2(i) \sigma \left( \sum_{\tau} W_1(i, \tau) x(t - \tau) \right).$$

Recall Lemma 12, each channel can be represented in the form of Volterra convolution, and this network can also be approximated by the sum of $M$ Volterra convolutions, the order-$n$ term is

$$\sum_{n=0} \mathbf{F}_{0,n} * \mathbf{x}^n + \sum_{n=0} \mathbf{F}_{1,n} * \mathbf{x}^n + \cdots + \sum_{n=0} \mathbf{F}_{M-1,n} * \mathbf{x}^n,$$

where $\mathbf{F}_{i,n},\ i = 0, 1, \cdots, M-1$; is the proxy kernel of each channel.

Based on the previous independent assumptions, these proxy kernels of different channels are also independent and identically. Recall Hoeffding's inequality (Vershynin, 2018), we have

$$\mathbb{P}\left\{\left|\frac{1}{M}\sum_{i=0}^{M-1} \mathbf{F}_{i,n} - \mu_n\right| \geq \eta\right\} \leq 2\exp\left(\frac{-2M\eta^2}{(b-a)^2}\right),$$

where $a, b$ are the minimum and maximum values of all proxy kernels, $a \leq F_{i,n}(\cdots) \leq b$. We can observe that for any $\epsilon \in (0, 1)$, $1/M \sum_{i=0}^{M-1} \mathbf{F}_{i,n}$ will converge to $\mu_n$, with probability at least $1 - \epsilon$ as long as

$$M \geq \frac{1}{2\eta^2} \ln\left(\frac{2}{\epsilon}\right)(b-a)^2. \tag{24}$$

### 3.1.2 Other Structures

Some commonly used structures can also be represented in the form of Volterra convolution, including some activations, normalize layers, inception modules, residual connection and pooling layers.

*ReLU activation.* The ReLU activation, $\max(x, 0)$ (Nair and Hinton, 2010), is not differentiable at point 0. This is quite a panic when taking Taylor expansion at that position. Nonetheless, it can be approximated by

$$\mathrm{ReLU}(x) = \lim_{\alpha\to\infty} \frac{1}{\alpha} \ln\left(1 + e^{\alpha x}\right).$$

Other activations of the ReLU family can be approximated in the same way.

*Fully connected layers.* A fully connected layer is a matrix multiplication with bias. It can be thought as discrete convolution with equal kernel length and signal length.

$$\sum_j W(i,j)x(j) + c = \sum_j \overline{W}(i, 0-j)x(j) + c,$$

where $\overline{W}(i, 0-j) = W(i, j)$.

*Normalization.* The normalization layer (Ioffe and Szegedy, 2015; Ba et al., 2016) scales and shifts the input signal

$$\mathbf{x} \to a\mathbf{x} + b.$$

This is a linear transformation, and this will not change the generality. Nevertheless, $a$ and $b$ are input related, which implies that the corresponded proxy kernels are also input related. In other words, the Volterra convolution is dynamic and will be updated if input differs. We will pause here and left this dynamic Volterra convolution to future work.

*Inception.* Main idea of inception module (Szegedy et al., 2015) is to apply convolution to different sizes of kernels parallelly and then concatenate, which is

$$\mathbf{g} * (\mathbf{h}_1 * \mathbf{x} + \mathbf{h}_2 * \mathbf{x} + \cdots + \mathbf{h}_n * \mathbf{x}) .$$

If we zero pad those kernels to the same size, recalling Property 7, it becomes a convolutional layer,

$$(\mathbf{g} \circledast (\mathbf{h}_1 + \mathbf{h}_2 + \cdots + \mathbf{h}_n)) * \mathbf{x}.$$

*Residual connection.* A residual connection (He et al., 2016) proposes $f(\mathbf{x}) + \mathbf{x}$, where $f(\cdot)$ is a neural network. If $f(\cdot)$ can be transformed to Volterra convolution, we have

$$\begin{aligned} f(\mathbf{x}) + \mathbf{x} &= \sum_{n=0}^{N} \mathbf{H}_n * \mathbf{x}^n + \mathbf{x} \\ &= \mathbf{H}_0 + (\mathbf{H}_1 * \mathbf{x} + \delta * \mathbf{x}) + \sum_{n=2}^{N} \mathbf{H}_n * \mathbf{x}^n \\ &= \mathbf{H}_0 + (\mathbf{H}_1 + \delta) * \mathbf{x} + \sum_{n=2}^{N} \mathbf{H}_n * \mathbf{x}^n, \end{aligned}$$

where $\delta$ is the Dirac delta and $\mathbf{H}_1 + \delta = \begin{cases} H_1(t) + 1, & t = 0 \\ H_1(t), & t \neq 0 \end{cases}$.

*Pooling.* Another family is the pooling layers. They are down sample or up sample operations. Average pooling is convolution with kernel filled by one. Max pooling picks the maximum value in a small region. It is data dependent, the equivalent kernels changes synchronously with input signal, and the proxy kernels are also dynamic.

### 3.2 Conversion of Layer Combination

In the previous subsection, we focused on small structures, and in this section, we will combine Volterra convolution layers, showing that the combinations also have the same format.

#### 3.2.1 Order-Two-Order-Two Structure

Before going further, a simple structure is presented in this subsection. The "order-2 — order-2" structure is stacking two order-two Volterra convolutions.

**Lemma 13** *The "order-2 — order-2" structure can be converted to the form of order-4 Volterra convolution.*

**Proof** Suppose $\mathbf{H}_i, \mathbf{G}_j$ are two groups of kernels, $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are the signals, these two Volterra convolutions are

$$\mathbf{y} = \sum_{i=0}^{2} \mathbf{H}_i * \mathbf{x}^i, \quad \mathbf{z} = \sum_{j=0}^{2} \mathbf{G}_j * \mathbf{y}^j.$$

Combining these two convolutions, we have

$$\mathbf{z} = \sum_{j=0}^{2} \mathbf{G}_j * \left(\mathbf{H}_0 + \mathbf{H}_1 * \mathbf{x}^1 + \mathbf{H}_2 * \mathbf{x}^2\right)^j .$$

The first term is $\mathbf{G}_0 * \mathbf{y}^0 = \mathbf{G}_0$.
Recall Property 1 and 7, the second term is

$$\begin{aligned}\mathbf{G}_1 * \mathbf{y}^1 &= \mathbf{H}_0 \sum \mathbf{G}_1 + \mathbf{G}_1 * (\mathbf{H}_1 * \mathbf{x}) + \mathbf{G}_1 * (\mathbf{H}_2 * \mathbf{x}^2) \\ &= \mathbf{H}_0 \sum \mathbf{G}_1 + (\mathbf{G}_1 \circledast \mathbf{H}_1) * \mathbf{x} + (\mathbf{G}_1 \circledast \mathbf{H}_2) * \mathbf{x}^2.\end{aligned}$$

Recall Property 4, the last term is

$$\begin{aligned}\mathbf{G}_2 * \mathbf{y}^2 &= \mathbf{G}_2 * \left(\mathbf{H}_0 + \mathbf{H}_1 * \mathbf{x} + \mathbf{H}_2 * \mathbf{x}^2\right)^2 \\ &= \mathbf{G}_2 * \lceil \mathbf{H}_0, \mathbf{H}_0 \rfloor + \mathbf{G}_2 * \lceil \mathbf{H}_0, \mathbf{H}_1 * \mathbf{x} \rfloor + \mathbf{G}_2 * \lceil \mathbf{H}_1 * \mathbf{x}, \mathbf{H}_0 \rfloor \\ &\quad + \mathbf{G}_2 * \left\lceil \mathbf{H}_0, \mathbf{H}_2 * \mathbf{x}^2 \right\rfloor + \mathbf{G}_2 * \left\lceil \mathbf{H}_2 * \mathbf{x}^2, \mathbf{H}_0 \right\rfloor + \mathbf{G}_2 * \lceil \mathbf{H}_1 * \mathbf{x}, \mathbf{H}_1 * \mathbf{x} \rfloor \\ &\quad + \mathbf{G}_2 * \left\lceil \mathbf{H}_1 * \mathbf{x}, \mathbf{H}_2 * \mathbf{x}^2 \right\rfloor + \mathbf{G}_2 * \left\lceil \mathbf{H}_2 * \mathbf{x}^2, \mathbf{H}_1 * \mathbf{x} \right\rfloor \\ &\quad + \mathbf{G}_2 * \left\lceil \mathbf{H}_2 * \mathbf{x}^2, \mathbf{H}_2 * \mathbf{x}^2 \right\rfloor .\end{aligned}$$

And recall Property 7, 9, 12, 13, 14, it becomes

$$\begin{aligned}\mathbf{G}_2 * \mathbf{y}^2 = \mathbf{H}_0^2 \sum \mathbf{G}_2 &+ \left(\sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_0, \mathbf{H}_1 \rfloor\right) * \mathbf{x} + \left(\sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_1, \mathbf{H}_0 \rfloor\right) * \mathbf{x} \\ &+ (\mathbf{G}_2 \circledast \lceil \mathbf{H}_1, \mathbf{H}_1 \rfloor) * \mathbf{x}^2 + \left(\sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_0, \mathbf{H}_2 \rfloor\right) * \mathbf{x}^2 + \left(\sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_2, \mathbf{H}_0 \rfloor\right) * \mathbf{x}^2 \\ &+ (\mathbf{G}_2 \circledast \lceil \mathbf{H}_2, \mathbf{H}_1 \rfloor) * \mathbf{x}^3 + (\mathbf{G}_2 \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor) * \mathbf{x}^3 + (\mathbf{G}_2 \circledast \lceil \mathbf{H}_2, \mathbf{H}_2 \rfloor) * \mathbf{x}^4.\end{aligned}$$

In summary, we have

$$\mathbf{z} = \sum_{k=0}^{4} \mathbf{F}_k * \mathbf{x}^k,$$

where the new kernels are

$$\begin{aligned}\mathbf{F}_0 &= \mathbf{G}_0 + \mathbf{H}_0 \sum \mathbf{G}_1 + \mathbf{H}_0^2 \sum \mathbf{G}_2 \\ \mathbf{F}_1 &= \mathbf{G}_1 \circledast \mathbf{H}_1 + \sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_0, \mathbf{H}_1 \rfloor + \sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_1, \mathbf{H}_0 \rfloor \\ \mathbf{F}_2 &= \mathbf{G}_1 \circledast \mathbf{H}_2 + \mathbf{G}_2 \circledast \lceil \mathbf{H}_1, \mathbf{H}_1 \rfloor + \sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_0, \mathbf{H}_2 \rfloor + \sum_{\#\mathbf{H}_0} \mathbf{G}_2 \circledast \lceil \mathbf{H}_2, \mathbf{H}_0 \rfloor \\ \mathbf{F}_3 &= \mathbf{G}_2 \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor + \mathbf{G}_2 \circledast \lceil \mathbf{H}_2, \mathbf{H}_1 \rfloor \\ \mathbf{F}_4 &= \mathbf{G}_2 \circledast \lceil \mathbf{H}_2, \mathbf{H}_2 \rfloor .\end{aligned} \tag{25}$$

■

Notice that it is possible for some $i$ such that $\|\mathbf{F}_i * \mathbf{x}^i\| = 0$, but we will also call it the form of order-four Volterra convolution.

### 3.2.2 Order-n-order-m Structure

In the following, we generalize "order-2 — order-2" structure (Lemma 13) to "order-$n$ — order-$m$" structure. This structure is to stack order-$n$ Volterra convolution and order-$m$ Volterra convolution.

**Lemma 14** *The "order-n — order-m" structure with $n, m > 0$ can be converted to the form of order-nm Volterra convolution.*

**Proof** Suppose $\mathbf{H}_i, \mathbf{G}_j$ are two groups of kernels, $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are signals, these two Volterra convolutions have the form of

$$\mathbf{y} = \sum_{i=0}^{n} \mathbf{H}_i * \mathbf{x}^i, \qquad \mathbf{z} = \sum_{j=0}^{m} \mathbf{G}_j * \mathbf{y}^j.$$

Combining these two Volterra convolutions, we have

$$\mathbf{z} = \sum_{j=0}^{m} \mathbf{G}_j * \left( \sum_{i=0}^{n} \mathbf{H}_i * \mathbf{x}^i \right)^j.$$

Recall Property 6, we have

$$\mathbf{z} = \sum_{j=0}^{m} \sum \binom{j}{j_0 j_1 \cdots j_n} \mathbf{G}_j * \left\lceil (\mathbf{H}_0 * \mathbf{x}^0)^{j_0}, (\mathbf{H}_1 * \mathbf{x}^1)^{j_1}, \cdots, (\mathbf{H}_n * \mathbf{x}^n)^{j_n} \right\rfloor,$$

where $\binom{j}{j_0 j_1 \cdots j_n} = \dfrac{j!}{j_0! j_1! \cdots j_n!}$ is multinomial coefficient and $\sum_{k=0}^{n} j_k = j, j_k \geq 0$, for all $k = 0, 1, \cdots, j$.

Recall Property 10, we have

$$\begin{aligned}
&\mathbf{G}_j * \left\lceil (\mathbf{H}_0 * \mathbf{x}^0)^{j_0}, (\mathbf{H}_1 * \mathbf{x}^1)^{j_1}, \cdots, (\mathbf{H}_n * \mathbf{x}^n)^{j_n} \right\rfloor \\
&= \left( \mathbf{G}_j \circledast \left\lceil \mathbf{H}_0^{j_0}, \mathbf{H}_1^{j_1}, \cdots, \mathbf{H}_n^{j_n} \right\rfloor \right) * \left\lceil \mathbf{x}^0, \mathbf{x}^{j_1}, \mathbf{x}^{2j_2} \cdots, \mathbf{x}^{nj_n} \right\rfloor \\
&= \left( \mathbf{G}_j \circledast \left\lceil \mathbf{H}_0^{j_0}, \mathbf{H}_1^{j_1}, \cdots, \mathbf{H}_n^{j_n} \right\rfloor \right) * \mathbf{x}^{j_1 + 2j_2 + \cdots + nj_n}.
\end{aligned}$$

In conclusion, the combination is

$$\mathbf{z} = \sum_{j=0}^{m} \left( \sum \binom{j}{j_0 j_1 \cdots j_n} \left( \mathbf{G}_j \circledast \left\lceil \mathbf{H}_0^{j_0}, \mathbf{H}_1^{j_1}, \cdots, \mathbf{H}_n^{j_n} \right\rfloor \right) * \mathbf{x}^{j_1 + 2j_2 + \cdots + nj_n} \right). \tag{26}$$

Clearly, $j_1 + 2j_2 + \cdots nj_n$ is sequential values from 0 to $nm$, which implies that this structure can also be converted to the form of order-$nm$ Volterra convolution. ■

This conversion does not convince that all terms are non-zero. It is possible that the $nm$ term is zero, $\|\mathbf{F}_{nm} * \mathbf{x}^{nm}\| = 0$, where $\mathbf{F}_{nm}$ is the proxy kernel. To keep the coherent, we would prefer to call it the form of order-$nm$ Volterra convolution.

Apart from the order, we also care about the number of terms, $\mathbf{G}_j \circledast \left[\mathbf{H}_0^{j_0}, \mathbf{H}_1^{j_1}, \cdots, \mathbf{H}_n^{j_n}\right]$, that added to a proxy kernel. For a given order $o, 0 \leq p \leq nm$, how many combinations of $j_0, j_1, \cdots, j_n, \sum_{k=0}^{n} j_k = 0, 1, \cdots, m$ such that $j_1 + 2j_2 + \cdots + nj_n = o$? This number can be obtained by counting the terms in Equation 26. We plot a figure for some combinations of $n, m$ in Figure 5.

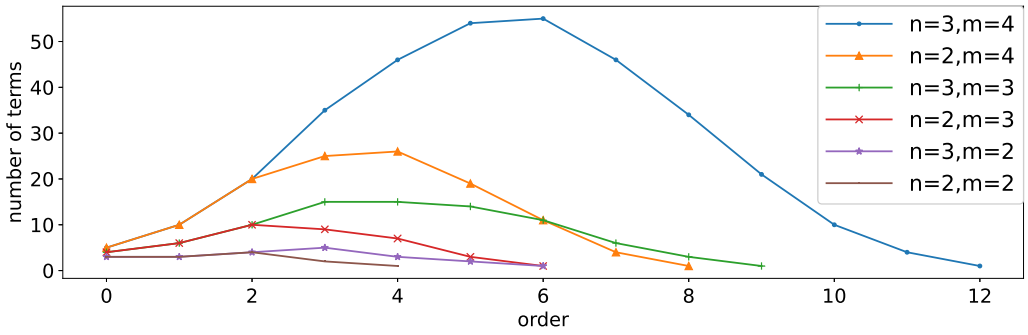



Figure 5: The number of terms that added to proxy kernels. The horizontal axis is the order and the vertical axis is the number of terms.

### 3.2.3 Multiple Channels and Layers

In the following, we show that multichannel and multi-layer structure can also be represented in the form of Volterra convolution. Moreover, we also measure the change of kernel size during this conversion.

**Lemma 15** *Multichannel convolution can be represented in the form of Volterra convolution.*

**Proof** Suppose that a multichannel convolution for signal $\mathbf{x}$ and kernel $\mathbf{h}$ is

$$y(c,t) = \sum_{u,\tau} h(c,u,\tau)x(u,t-\tau) = \sum_{u} h(c,u) * x(u).$$

It can be considered as sum of multiple convolutions, which have exactly the same form. ■

**Lemma 16** *Stacked order $o_1, o_2, \cdots, o_n$ Volterra convolutions can be converted to the form of order-$\prod_{i=1}^{n} o_i$ Volterra convolution, where $o_1, o_2, \cdots, o_n > 0$.*

**Proof** To prove this, we recursively apply Lemma 14. Taking the first two layers into consideration, we have order-$(o_1 o_2)$ Volterra convolution. Appending the third layer, we have order-$(o_1 o_2 o_3)$ Volterra convolution. Recursively, after appending the $n$-th layer, the order becomes $\prod_{i=1}^{k+1} o_i$, which completes this proof. ■

**Remark 17** *With number of stacked layers increases, the order of converted Volterra convolution grows exponentially.*

Previous lemmas show us the change of orders. The following lemma will measure the change of kernel size. This lemma will be helpful if we want to compute the overall sizes, strides, and paddings of proxy kernels.

**Lemma 18** *Suppose that size of kernels are $z_1, z_2, \cdots, z_n$, strides are $s_1, s_2, \cdots, s_n$, and paddings are $p_1, p_2, \cdots, p_n$, size of combined kernel is $z_1+(z_2-1)s_1+(z_3-1)s_1s_2+\cdots+(z_n-1)\prod_{k=1}^{n-1} s_k$, combined stride is $\prod_{k=1}^{n} s_k$ and combined padding is $p_1+p_2s_1+\cdots+p_n\prod_{k=1}^{n-1} s_k$.*

**Proof** Size of convolution is [2]

$$\text{out\_size} = \frac{\text{in\_size} + 2\ \text{padding} - \text{kernel\_size}}{\text{stride}} + 1.$$

Stacking the first two layers, we have

$$\begin{aligned}\text{out\_size} &= \frac{\left(\text{in\_size} + 2p_1 - z_1\right)/s_1 + 1 + 2p_2 - z_2}{s_2} + 1 \\ &= \frac{\text{in\_size} + 2(p_1 + p_2 s_1) - (z_1 + (z_2 - 1)s_1)}{s_1 s_2} + 1.\end{aligned}$$

Equivalent size is $z_1 + (z_2 - 1)s_1$, stride is $s_1s_2$, and padding is $p_1 + p_2s_1$. Stacking the first three layers, we have

$$\text{out\_size} = \frac{\text{in\_size} + 2(p_1 + p_2s_1 + p_3s_1s_2) - (z_1 + (z_2 - 1)s_1 + (z_3 - 1)s_1s_2)}{s_1s_2s_3} + 1.$$

Equivalent size is $z_1+(z_2-1)s_1+(z_3-1)s_1s_2$, stride is $s_1s_2s_3$, and padding is $p_1+p_2s_1+p_3s_1s_2$. Recursively, this proof is completed. ■

### 3.3 Validation of Lemma 12 and Lemma 13

In this subsection, we validate the approximation of two simple structures (truncated version of Lemma 12 and Lemma 13), which are two fundamental structures in our proof and which can show the effectiveness and correctness of our results.

The set $\mathbb{M}$ is defined for numerical validation in this subsection and below,

$$\mathbb{M} = \left\{ \mathbf{x} : \mathbf{x} = \frac{\mathbf{y}}{\|\mathbf{y}\|_2}, y(\cdot) \sim \mathcal{N}(0,1) \right\}, \tag{27}$$

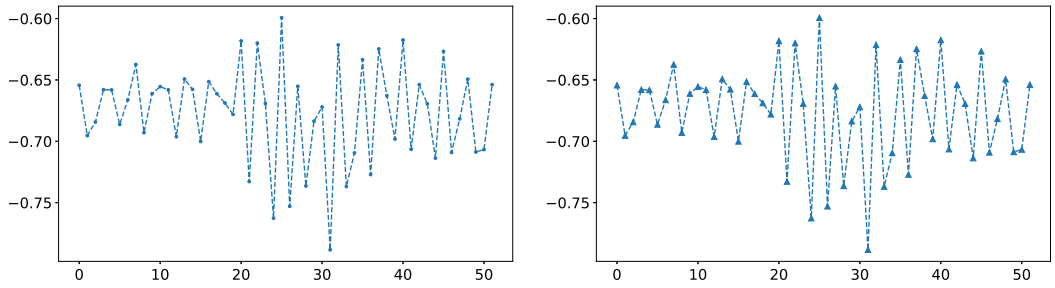
where $\mathcal{N}(0,1)$ is standard Gaussian distribution with mean zero and variance one and $\|\mathbf{y}\|_2$ is $L_2$ norm of $\mathbf{y}$. The upper script of these symbols indicates the shape, i.e., $\mathbf{H} \in \mathbb{M}^{9\times 9}$ is a matrix taken from $\mathbb{M}$ and the shape is $(9, 9)$.

2. https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html

*Validate Lemma 12.* We will check whether the "conv — act — conv" structure can be approximated in the form of Volterra convolution. Take the first four terms into consideration and activation function $\sigma(t) = 1/(1+e^{-t})$, we have

$$\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x}) \approx \frac{1}{2}\sum \mathbf{g} + \frac{1}{4}(\mathbf{g} \circledast \mathbf{h}) * \mathbf{x} - \frac{(\mathrm{diag}(3, \mathbf{g}) \circledast \mathbf{h}^3) * \mathbf{x}^3}{48} + \frac{(\mathrm{diag}(5, \mathbf{g}) \circledast \mathbf{h}^5) * \mathbf{x}^5}{480}. \quad (28)$$

This approximation relies on the Taylor expansion, which is valid only in a small neighbor of zero. Therefore, we need to make sure that $\mathbf{h} * \mathbf{x}$ is located in such neighbor. We randomly generate $\mathbf{x} \in \mathbb{M}^{64}$, $\mathbf{h} \in \mathbb{M}^9$ and $\mathbf{g} \in \mathbb{M}^5$. Convolution with sigmoid activation is plotted in Figure 6a and the approximated order-five Volterra convolution is plotted in Figure 6b.




(a) $\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})$ (b) $\frac{1}{2}\sum \mathbf{g} + \frac{1}{4}(\mathbf{g} \circledast \mathbf{h}) * \mathbf{x} + \cdots$

Figure 6: (6a) The output of convolution with activation (left-hand side of Equation 28); (6b) The output of the approximated Volterra convolution (right-hand side of Equation 28). The reconstruct error ($L_2$-norm) between left and right is $5.64877e^{-07}$.

*Validate Lemma 13.* We will check whether the "order-2 — order-2" structure can be converted to the form of order-four Volterra convolution,

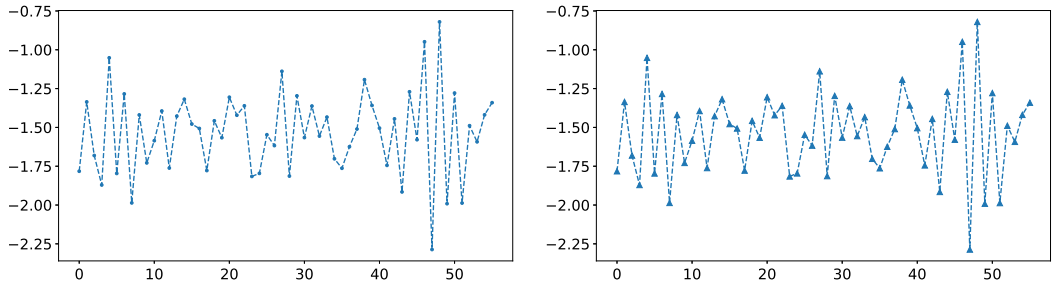
$$\sum_{i=0}^{2} \mathbf{G}_i * \left( \sum_{j=0}^{2} \mathbf{H}_j * \mathbf{x}^j \right)^i = \sum_{k=0}^{4} \mathbf{F}_k * \mathbf{x}^k, \quad (29)$$

where $\mathbf{F}_k$ are kernels from Equation 25.

We randomly generate $H_0, G_0 \sim \mathcal{N}(0,1)$, $\mathbf{H}_1, \mathbf{G}_1 \in \mathbb{M}^5$ and $\mathbf{H}_2, \mathbf{G}_2 \in \mathbb{M}^{5\times 5}$, and $\mathbf{x} \in \mathbb{M}^{64}$. The left-hand side of Equation 29 is plotted in Figure 7a and the right-hand side is plotted in Figure 7b. It shows that these two are exactly the same.

## 4. Inferring the Proxy Kernels

All we need to approximate a convolutional neural network to finite term Volterra convolution are the proxy kernels. If the structures and all parameters of a well-trained network is known, i.e., white box, the proxy kernels can be explicitly computed, just as demonstrated in Subsection 3.3. However, in some cases, the structure or parameters are not accessible and only the input output pairs are known, i.e., the network is a black box. This may happen when we want to approximate a network using black-box mode. In such cases, the proxy




(a) $\sum_{i=0}^{2} \mathbf{G}_i * \left(\sum_{j=0}^{2} \mathbf{H}_j * \mathbf{x}^j\right)^i$ (b) $\sum_{k=0}^{4} \mathbf{F}_k * \mathbf{x}^k$

Figure 7: (7a) The output of stacking two order-two Volterra convolutions (left-hand side of Equation 29); (7b) The output of order-four Volterra convolutions (right-hand side of Equation 29). The reconstruct error ($L^2$-norm) between left and right is $1.79705e^{-15}$.

kernels can be approximated by training a hacking network, which will be described in detail below.

Basically, a hacking network is a neural network that has the structure of finite term Volterra convolution. The number of terms of the hacking network controls the order of Volterra convolution and hence the approximation precision. To train the hacking network, we feed the input-output pairs generated by target network (or its parts) to the hacking network, and minimize the mean square error between the output of the hacking network and that of the target network.

To validate the effectiveness of hacking network, we build a network and compute its order-zero and order-one proxy kernels manually. Then, we train a hacking network to infer these kernels and compare whether they are the same. To infer the order-zero and order-one terms, we need to build a hacking network with the structure of order-one Volterra convolution, i.e., $\mathbf{w} * \mathbf{x} + b$. This structure can be implemented by a single convolutional layer. If the hacking network is well-trained, the order-zero proxy kernel is the bias of the convolutional layer, and the order-one proxy kernel is the weight.

### 4.1 Two Examples

As our first example, we consider a simple two-layer network with sigmoid activation. Suppose that a two-layer network is represented as $\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})$. Without loss of generality, we use the same settings as Equation 28,

$$\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x}) \approx \frac{1}{2}\sum \mathbf{g} + \frac{1}{4}(\mathbf{g} \circledast \mathbf{h}) * \mathbf{x} - \frac{(\mathrm{diag}(3, \mathbf{g}) \circledast \mathbf{h}^3) * \mathbf{x}^3}{48} + \frac{(\mathrm{diag}(5, \mathbf{g}) \circledast \mathbf{h}^5) * \mathbf{x}^5}{480}.$$

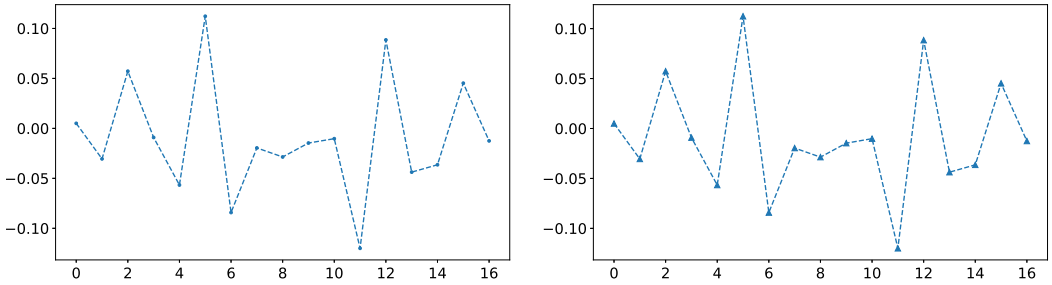
It is clear that, the order-zero proxy kernel is $\frac{1}{2}\sum \mathbf{g}$, and the order-one proxy kernel is $\frac{1}{4}(\mathbf{g} \circledast \mathbf{h})$.

We randomly pick two kernels,

$$\mathbf{h} = \begin{bmatrix} 0.0381 & -0.2047 & 0.3097 & 0.0693 & -0.3405 & 0.7618 & -0.1190 & -0.1089 & 0.3657 \end{bmatrix};$$
$$\mathbf{g} = \begin{bmatrix} 0.5280 & -0.3684 & -0.2644 & -0.3412 & -0.2461 & -0.1377 & -0.3077 & 0.4540 & -0.1369 \end{bmatrix}.$$

The hacking network is a single layer convolutional network. While training, we randomly generate the input data $\mathbf{x} \in \mathbb{M}^{\text{batch\_size}\times 1\times \text{length}}$ and minimize mean square error of the output of the hacking network and the output of $\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})$.

By straightforward calculation, the order-zero proxy kernel is $-4.10236e^{-01}$, and that obtained from the hacking network is $-4.10295e^{-01}$, and the inferred order-one proxy kernels respectively obtained by direct calculation and by training the hacking network are compared in Figure 8. In summary, the hacking network can approximate the true order-zero and order-one proxy kernels very well.




(a) $\frac{1}{4}(\mathbf{g} \circledast \mathbf{h})$ (b) kernel from the trained hacking network

Figure 8: The inferred order-one proxy kernels from the two-layer network. The reconstruction error ($L_2$-norm) between left and right is $3.04194e^{-04}$.

For the second example, we consider a three-layer network with sigmoid activation. Suppose that a three-layer network is represented as $\mathbf{f} * \sigma(\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x}))$. Using the same settings as previous example and approximating this three-layer network into the form of Volterra convolution, we have

$$\begin{aligned}
&\mathbf{f} * \sigma(\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})) \\
&= \frac{1}{2}\sum \mathbf{f} + \frac{1}{4}\mathbf{f} * [\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})] - \frac{1}{48}\mathbf{f} * [\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})]^3 + \frac{1}{480}\mathbf{f} * [\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})]^5 \\
&= \frac{1}{2}\sum \mathbf{f} + \frac{1}{4}\mathbf{f} * (\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x})) - \frac{1}{48}\text{diag}(3, \mathbf{f}) * (\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x}))^3 \\
&\quad + \frac{1}{480}\text{diag}(5, \mathbf{f}) * (\mathbf{g} * \sigma(\mathbf{h} * \mathbf{x}))^5. \qquad \text{(Property 15)}
\end{aligned}$$

Substituting Equation 28 and reordering the terms, we obtain the order-zero proxy kernel,

$$\left(\sum \mathbf{f}\right)\left(\frac{1}{2} + \frac{1}{4}\left(\frac{1}{2}\sum \mathbf{g}\right) - \frac{1}{48}\left(\frac{1}{2}\sum \mathbf{g}\right)^3 + \frac{1}{480}\left(\frac{1}{2}\sum \mathbf{g}\right)^5\right),$$

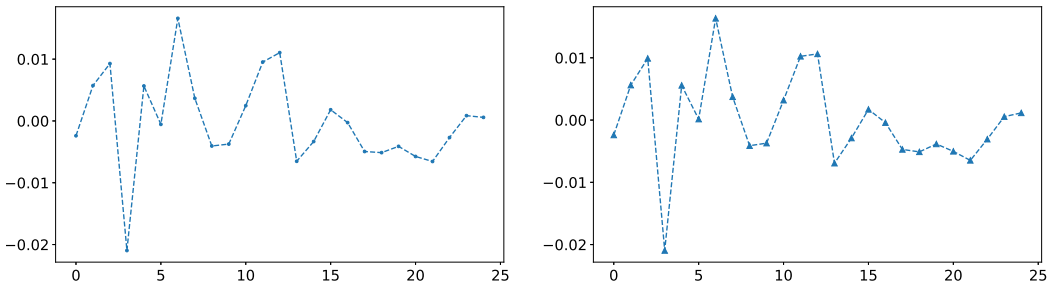
and the order-one proxy kernel,

$$\frac{1}{4}\left(\frac{1}{4} - \frac{3}{48}\left(\frac{1}{2}\sum \mathbf{g}\right)^2 + \frac{5}{480}\left(\frac{1}{2}\sum \mathbf{g}\right)^4\right)(\mathbf{f} \circledast \mathbf{g} \circledast \mathbf{h}). \tag{30}$$

We randomly pick three kernels,

$$\begin{aligned}
\mathbf{h} &= \begin{bmatrix} 0.4830 & -0.3142 & -0.2219 & 0.0361 & 0.2659 & 0.4040 & 0.3978 & 0.3628 & 0.3061 \end{bmatrix}; \\
\mathbf{g} &= \begin{bmatrix} -0.1271 & 0.1521 & 0.6264 & -0.2576 & 0.3027 & -0.1574 & 0.1009 & 0.3923 & 0.4705 \end{bmatrix}; \\
\mathbf{f} &= \begin{bmatrix} 0.7160 & -0.3866 & -0.1870 & -0.0566 & 0.3057 & -0.0062 & -0.4463 & -0.0395 & 0.0735 \end{bmatrix}.
\end{aligned}$$

The training process of the hacking network is the same as the previous example. The order-zero proxy kernel by direct calculation is $-1.83091e^{-02}$, and that by training the hacking network is $-1.82950e^{-02}$, and the order-one proxy kernels obtained by these two ways shown in Figure 9. The figure shows again that the proxy kernels obtained training the hacking network is almost the same as that obtained by direct calculation.




(a) kernel from Equation 30 (b) kernel from the trained hacking network

Figure 9: The inferred order-one proxy kernels of the three-layer network. The reconstruction error ($L_2$-norm) between left and right is $1.94667e^{-03}$.

These two examples show that the order-zero and order-one proxy kernels inferred by direct calculation and by training the hacking network are very close to each other. To cover a broader selection of parameters, we analyze the statistics of the reconstruction errors of the proxy kernels inferred from these two methods. The results are illustrated in Figure 10. It shows that the choice of parameters has a less effect on the reconstruction error and these two methods are comparable.

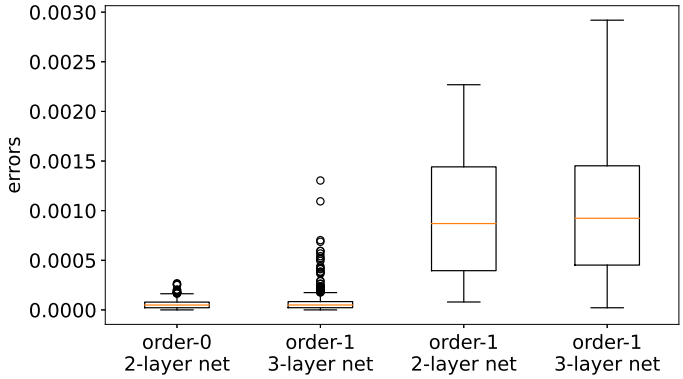



Figure 10: The boxplot of reconstruction errors between two methods.

### 4.2 An Application of the Order-one Proxy Kernels

Why do we need to infer the proxy kernels?

We think the proxy kernels shall contain some useful information about the original network. It is possible to carefully design special inputs according to the inferred proxy kernels to change the behavior of the original network. Below, we provide a toy example to illustrate this interesting application of order-one proxy kernels. First, we build a hacking network to approximate the order-one proxy kernel of a classifier network trained on the MNIST data set (Lecun et al., 1998). Then, we add visually imperceptible perturbations to the inputs for cheating the classifier to give wrong labels.

The structure of the classifier network is shown in the left of Table 1. Training images are all scaled to $[0, 1]$. To obtain a network that is robust to noise, uniform noise $\mathcal{U}(0, 0.2)$ and Gaussian noise $\mathcal{N}(0, 0.2)$ are respectively added to training images with a probability of 5%. After 512 training episodes, the network achieved 98.280% accuracy on test set.

The hacking network is illustrated in the right of Table 1. We only approximate the first six layers of the classifier network. This is because approximating the entire classifier network to a linear network will result in high fitting errors, and these errors will make it harder for hacking network to converge. From Table 1, the hacking network has ten output channels. Hence, we have ten proxy kernels $\mathbf{h}_1, \cdots, \mathbf{h}_{10}$, corresponding to the ten output channels.

| | classifier network | hacking network |
|---|---|---|
| | input | input |
| 1 | conv2d(1, 10, k=3, s=2, p=1) | conv2d(1, 10, k=15, s=8, p=3) |
| 2 | sigmoid | |
| 3 | conv2d(10, 10, k=3, s=2, p=1) | |
| 4 | sigmoid | |
| 5 | conv2d(10, 10, k=3, s=2, p=0) | |
| 6 | sigmoid | |
| 7 | conv2d(10, 10, k=3, s=1, p=0) | output |
| 8 | sigmoid | |
| 9 | flatten, linear(10, 10) | |
| | output | |

Table 1: Structure of classifier network and hacking network. Parameters of conv2d layers are input channels, output channels, kernel size, stride, and padding.

The training of the hacking network is similar to the procedure discussed in the previous subsection. The training images and their outputs of the classifier network from input-label pairs, and the hacking network is trained with these pairs until convergence. Compared to the outputs of the first six layers of the classifier network, the mean square error of the hacking network is about $3.19016e^{-02}$. In other words, these two networks have the similar behavior.

Suppose that some neurons in the network are suppressed when feeding input image $\mathbf{x}$. To change the behavior of this network, we need to activate some extra neurons. Specifically, in this application, we try to increase the energy of these extra neurons to activate them.

One important advantage of the hacking network is its simple structure: only convolutions are involved. Hence, to largely change the outputs, for given order-one proxy kernel $\mathbf{h}$ and input signal $\mathbf{x}$, we search for perturbation $\epsilon$ such that

$$\underset{\epsilon}{\operatorname{argmax}} \|\mathbf{h} * (\mathbf{x} + \epsilon)\|_2, \quad \text{s.t. } \|\epsilon\|_2 \leq c, \tag{31}$$

where $c$ is a constant. If $\mathbf{h}$ and $\mathbf{x} + \epsilon$ are close to each other in the frequency domain, the energy $\|\mathbf{h} * (\mathbf{x} + \epsilon)\|_2$ should be large.

Recall Parseval's Theorem and Convolution Theorem, convolution in time domain equals multiplication in frequency domain and the energy is preserved, $\epsilon$ can be obtained via Fourier transform $\mathcal{F}(\cdot)$,

$$\mathcal{F}(\epsilon) \propto \mathcal{F}(\mathbf{h}) - \mathcal{F}(\mathbf{x}). \tag{32}$$

Applying the bounded condition, we scale it by $\alpha$ and take inverse Fourier transform $\mathcal{F}^{-1}(\cdot)$,

$$\epsilon = \alpha \mathcal{F}^{-1} \left( \mathcal{F}(\mathbf{h}) - \mathcal{F}(\mathbf{x}) \right). \tag{33}$$

Recall Equation 33, we can compute a perturbation as

$$\epsilon_i = \mathcal{F}^{-1} \left( \mathcal{F}(\mathbf{w}_i) - \mathcal{F}(\mathbf{x}) \right).$$

Since each pixel of the input image is in the range of $[0, 1]$, the perturbation should be adjusted to the same range. In this application, to make the perturbation hard to be seen, $\epsilon_i$ is scaled by a factor $\alpha$. The value of $\alpha$ is 0.55 in the experiment below. In short, the perturbation $\epsilon_i$ is normalized as

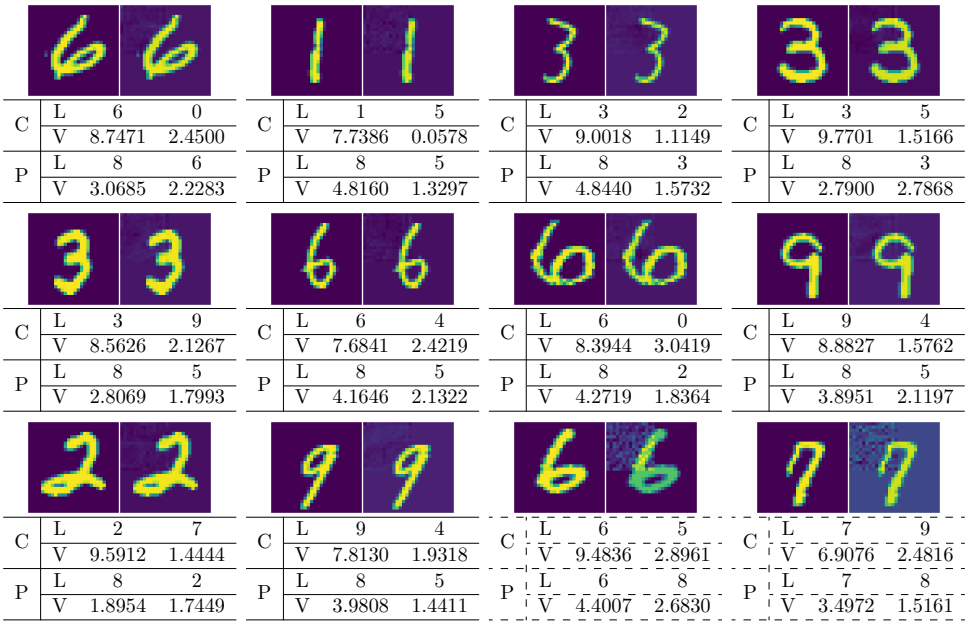
$$\epsilon_i \leftarrow \alpha \frac{\epsilon_i - \min(\epsilon_i)}{\max(\epsilon_i) - \min(\epsilon_i)}.$$

Besides the ten proxy kernels estimated by the hacking network, we also append two fake kernels to check whether the fake kernels can behave similarly. The fake kernels are $\mathbf{h}_{11} \sim \mathcal{U}(0, 1)$ and $\mathbf{h}_{12} \sim \mathcal{N}(0, 1)$. We randomly choose twelve images from the test set, $\mathbf{x}_i, i = 1, 2, \cdots, 12$, and feed both image $\mathbf{x}_i$ and $\mathbf{x}_i + \epsilon_i$ into classifier network, where $\epsilon_1, \epsilon_2, \cdots, \epsilon_{10}$ are computed from the approximated order-one proxy kernels, and $\epsilon_{11}, \epsilon_{12}$ are computed from two fake kernels. Results are illustrated in Figure 11. The images with patches computed from the approximated order-one proxy kernels are more likely to change the output than the images with patches computed from fake kernels.

This phenomenon is similar to the adversarial example (Szegedy et al., 2014; Goodfellow et al., 2015), adding a human invisible perturbation to an image can mislead the network. Nevertheless, these two methods are different. They iteratively compute perturbations that mislead the network with or without specific objects. Our method is to maximize the energy of order-one convolution, and this increases the probability to change output of the network. More details about the perturbations are presented in Appendix C.

## 5. Discussion

This paper presents a new perspective on the analysis of convolutional neural networks. It shows that a convolutional neural network can be represented in the form of Volterra



| | | | |
|---|---|---|---|
| C | L | 6 | 0 |
| | V | 8.7471 | 2.4500 |
| P | L | 8 | 6 |
| | V | 3.0685 | 2.2283 |

| | | | |
|---|---|---|---|
| C | L | 1 | 5 |
| | V | 7.7386 | 0.0578 |
| P | L | 8 | 5 |
| | V | 4.8160 | 1.3297 |

| | | | |
|---|---|---|---|
| C | L | 3 | 2 |
| | V | 9.0018 | 1.1149 |
| P | L | 8 | 3 |
| | V | 4.8440 | 1.5732 |

| | | | |
|---|---|---|---|
| C | L | 3 | 5 |
| | V | 9.7701 | 1.5166 |
| P | L | 8 | 3 |
| | V | 2.7900 | 2.7868 |

| | | | |
|---|---|---|---|
| C | L | 3 | 9 |
| | V | 8.5626 | 2.1267 |
| P | L | 8 | 5 |
| | V | 2.8069 | 1.7993 |

| | | | |
|---|---|---|---|
| C | L | 6 | 4 |
| | V | 7.6841 | 2.4219 |
| P | L | 8 | 5 |
| | V | 4.1646 | 2.1322 |

| | | | |
|---|---|---|---|
| C | L | 6 | 0 |
| | V | 8.3944 | 3.0419 |
| P | L | 8 | 2 |
| | V | 4.2719 | 1.8364 |

| | | | |
|---|---|---|---|
| C | L | 9 | 4 |
| | V | 8.8827 | 1.5762 |
| P | L | 8 | 5 |
| | V | 3.8951 | 2.1197 |

| | | | |
|---|---|---|---|
| C | L | 2 | 7 |
| | V | 9.5912 | 1.4444 |
| P | L | 8 | 2 |
| | V | 1.8954 | 1.7449 |

| | | | |
|---|---|---|---|
| C | L | 9 | 4 |
| | V | 7.8130 | 1.9318 |
| P | L | 8 | 5 |
| | V | 3.9808 | 1.4411 |

| | | | |
|---|---|---|---|
| C | L | 6 | 5 |
| | V | 9.4836 | 2.8961 |
| P | L | 6 | 8 |
| | V | 4.4007 | 2.6830 |

| | | | |
|---|---|---|---|
| C | L | 7 | 9 |
| | V | 6.9076 | 2.4816 |
| P | L | 7 | 8 |
| | V | 3.4972 | 1.5161 |

Figure 11: Predictions of MNIST digits polluted by perturbations. For each frame, top left is the original image $\mathbf{x}_i$, top right is the patched image $\mathbf{x}_i + \epsilon_i$, and table below is prediction. Block "C" indicate top two predict values of $\mathbf{x}_i$, and block "P" indicate top two predict values of $\mathbf{x}_i + \epsilon_i$. Line "L" indicate labels and Line "V" indicate values.

convolution. By approximating infinite Volterra convolution using the finite one, we can analyze the proxy kernels of Volterra convolution rather than directly analyze the original network, even if the structure and the parameters of the network are unknown. The proxy kernels can be computed from the weights of a neural network, or be approximated by building and training a hacking network. The method of approximating the proxy kernels is more intuitive and easier to implement if the number of layers is large, and the approximated proxy kernels are comparable with the actual kernels.

In future work, we plan to do further research on the proxy kernels. For example, how to compute the high order kernels efficiently, how to determine the order of kernels with given approximation accuracy, and so on. We will also investigate the kernels that are input related, i.e., the dynamic networks, as well as practical applications of the hacking network.

All codes associated with this article are accessible publicly at GitHub `https://github.com/tenghuilee/nnvolterra.git`.

## Acknowledgments

The authors thank Dr. Andong Wang for helpful discussions. Sincere thanks to all anonymous reviewers, that greatly helped to improve this paper. This work was supported in part by the Natural Science Foundation of China under Grants 62073087, 62071132, 61973090, and 62203124, and JSPS KAKENHI Grant Number 20H04249, 20H04208.

## Appendix A. Convolution From Order One to Two

In Appendix A, we show how to convert a one-dimensional order-one convolution to a one-dimensional order-two convolution in continuous time domain only. By the well-known continuous one-dimensional convolution (Equation 3), and the differential property of one-dimensional convolution, we have

$$\begin{aligned}(\mathbf{h} * \mathbf{x})(t) &= \int_{-\infty}^{+\infty} h(\tau)x(t-\tau)d\tau \\ &= \int_{-\infty}^{+\infty} h(t-\tau)x(\tau)d\tau \\ &= \int_{-\infty}^{+\infty} \frac{dh(t-\tau)}{d\tau}\left(\int_{-\infty}^{\tau} x(l)dl\right)d\tau.\end{aligned}$$

Applying the integration, $\int_{-\infty}^{\tau} x(l)dl = 1/2x(\tau)x(\tau) - 0$, we have

$$\begin{aligned}(\mathbf{h} * \mathbf{x})(t) &= \int_{-\infty}^{+\infty} \frac{1}{2}\frac{dh(t-\tau)}{dt}x(\tau)x(\tau)d\tau \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \frac{1}{2}\frac{dh(t-\tau)}{dt}x(\tau)x(\iota)\delta(\tau-\iota)d\tau d\iota,\end{aligned}$$

where Dirac delta $\delta(t) = \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases}$. Let $g(t-\tau, t-\iota) = \frac{1}{2}\frac{dh(t-\tau)}{dt}\delta(\tau-\iota)$. We have

$$\begin{aligned}(\mathbf{h} * \mathbf{x})(t) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} g(t-\tau, t-\iota)x(\tau)x(\iota)d\tau d\iota \\ &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} g(\tau, \iota)x(t-\tau)x(t-\iota)d\tau d\iota \\ &= (\mathbf{g} * \lceil \mathbf{x}, \mathbf{x} \rfloor)(t).\end{aligned}$$

During this, we are surprised to find that $\mathbf{g}$ is insensitive to time irrelevant additive noise. Suppose $\hat{h}(\tau) = h(\tau) + n(\tau)$, with $\frac{dn(\tau)}{d\tau} = 0$, we have

$$g(t-\tau, t-\iota) = \frac{1}{2}\frac{d\hat{h}(t-\tau)}{dt}\delta(\tau-\iota) = \frac{1}{2}\frac{dh(t-\tau)}{d\tau}\delta(\tau-\iota).$$

## Appendix B. Proof for Combination Properties

In this appendix, only properties of discrete one-dimensional signals are proved, and this proof can be generalized to other situations. To prevent indexing out of bound, zero padding is always considered.

**Proof**

*Property 1 and 3.* They can be proved by linearity of integration or summation.

*Property 2.* Due to linear property of integration or summation, we can separate these two terms as

$$\mathbf{G} * (\vec{\mathbf{x}} + \alpha) = \mathbf{G} * \vec{\mathbf{x}} + \alpha \sum_{\vec{\mathbf{t}}} G(\vec{\mathbf{t}}).$$

*Property 4, 5 and 6.* We will prove Property 6. Noticing that $\mathbf{G}$ is symmetric, swapping order of $\mathbf{x}_i$ does not change the result.

$$
\begin{aligned}
&\left(\mathbf{G} * (\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_m)^n\right)(t) \\
&= \sum_{\tau_1,\cdots,\tau_n} G(\tau_1,\cdots,\tau_n) \prod_{i=1}^{n} (x_1(t-\tau_i) + x_2(t-\tau_i) + \cdots + x_m(t-\tau_i)) \\
&= \sum_{\tau_1,\cdots,\tau_n} \sum \binom{n}{n_1 n_2 \cdots n_m} G(\tau_1,\cdots,\tau_n) \\
&\qquad x_1(t-\tau_1)x_1(t-\tau_2)\cdots x_1(t-\tau_{n_1}) \\
&\qquad x_2(t-\tau_{1+n_1})x_2(t-\tau_{2+n_1})\cdots x_2(t-\tau_{n_2+n_1}) \\
&\qquad \cdots \\
&\qquad x_m(t-\tau_{1+\sum_{k=1}^{m-1} n_k})x_m(t-\tau_{2+\sum_{k=1}^{m-1} n_k})\cdots x_m(t-\tau_{n_m+\sum_{k=1}^{m-1} n_k}) \\
&= \left(\sum \binom{n}{n_1 n_2 \cdots n_m} \mathbf{G} * \lceil \mathbf{x}_1^{n_1}, \mathbf{x}_2^{n_2}, \cdots, \mathbf{x}_m^{n_m} \rfloor\right)(t),
\end{aligned}
$$

where $\binom{n}{n_1 n_2 \cdots n_m} = \dfrac{n!}{n_1! n_2! \cdots n_m!}$, the multinomial coefficient, and $\sum_{i=1}^{m} n_i = m, n_i \geq 0$, for all $i = 1, 2, \cdots, m$. If $m = 2$, this is Property 5, and if $m = 2$ and $n = 2$, this is Property 4.

*Property 7 and 8.* Only Property 8 is proved, and we can prove Property 7 by setting stride equal to one.

$$
\begin{aligned}
\left(\mathbf{G} *_s (\mathbf{H} *_z \vec{\mathbf{x}})\right)(t) &= \sum_{l} G(l) \sum_{\vec{\tau}} H(\vec{\tau}) \prod_{i=1}^{n} x_i(z(st-l) - \tau_i) \\
&= \sum_{\vec{\tau}} \left( \sum_{l} G(l) H(\vec{\tau} - zl) \right) \prod_{i=1}^{n} x_i(szt - \tau_i) \\
&= \sum_{\vec{\tau}} (\mathbf{G} \circledast_z \mathbf{H})(\vec{\tau}) \prod_{i=1}^{n} x_i(szt - \tau_i) \\
&= \left((\mathbf{G} \circledast_z \mathbf{H}) *_{sz} \vec{\mathbf{x}}\right)(t).
\end{aligned}
$$

*Property 9 and 10.* We will prove Property 9 here and this proof could be intuitively extended to Property 10.

$$
\begin{aligned}
&\left(\mathbf{G} * \lceil \mathbf{H}_1 * \vec{\mathbf{x}}, \mathbf{H}_2 * \vec{\mathbf{y}} \rfloor\right)(t) \\
&= \sum_{l_1,l_2} G(l_1,l_2) \left( \sum_{\vec{\tau}_1} H_1(\vec{\tau}_1) \prod_{i=1}^{n} x_i(t - \tau_{1,i} - l_1) \right) \left( \sum_{\vec{\tau}_2} H_2(\vec{\tau}_2) \prod_{j=1}^{m} y_j(t - \tau_{2,j} - l_2) \right) \\
&= \sum_{\vec{\tau}_1,\vec{\tau}_2} \left( \sum_{l_1,l_2} G(l_1,l_2) H_1(\vec{\tau}_1 - l_1) H_2(\vec{\tau}_2 - l_2) \right) \left( \prod_{i=1}^{n} x_i(t - \tau_{1,i}) \right) \left( \prod_{j=1}^{m} y_j(t - \tau_{2,j}) \right) \\
&= \sum_{\vec{\tau}_1,\vec{\tau}_2} (\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor)(\vec{\tau}_1, \vec{\tau}_2) \left( \prod_{i=1}^{n} x_i(t - \tau_{1,i}) \right) \left( \prod_{j=1}^{m} y_j(t - \tau_{2,j}) \right) \\
&= \left((\mathbf{G} \circledast \lceil \mathbf{H}_1, \mathbf{H}_2 \rfloor) * \lceil \vec{\mathbf{x}}, \vec{\mathbf{y}} \rfloor\right)(t).
\end{aligned}
$$

*Property 11.*

$$\begin{aligned}(\mathbf{G}_1 \circledast (\mathbf{G}_2 \circledast \mathbf{G}_3))(\vec{\tau}) &= \sum_k G_1(k) \left( \sum_l G_2(l) G_3(\vec{\tau} - k - l) \right) \\ &= \sum_l \left( \sum_k G_1(k) G_2(l-k) \right) G_3(\vec{\tau} - l) \\ &= ((\mathbf{G}_1 \circledast \mathbf{G}_2) \circledast \mathbf{G}_3)(\vec{\tau}).\end{aligned}$$

*Property 12, 13, and Property 14.* We will only prove Property 14 here, and we can prove Property 12 and 13 in the same way.

$$\begin{aligned}&(\mathbf{G} * \lceil \mathbf{H}_1 * \vec{\mathbf{x}}, \alpha, \mathbf{H}_2 * \vec{\mathbf{y}} \rfloor)(t) \\ &= \alpha \sum_{l_1,l_2,l_3} G(l_1,l_2,l_3) \left( \sum_{\vec{\tau}_1} H_1(\vec{\tau}_1) \prod_{i=1} x_i(t - \tau_{1,i} - l_1) \right) \left( \sum_{\vec{\tau}_3} H_2(\vec{\tau}_3) \prod_{i=1} y_i(t - \tau_{3,i} - l_3) \right) \\ &= \alpha \sum_{\vec{\tau}_1,\vec{\tau}_2} \left( \sum_{l_1,l_2,l_3} G(l_1,l_2,l_3) H_1(\vec{\tau}_1 - l_1) H_2(\vec{\tau}_3 - l_3) \right) \left( \prod_{i=1} x_i(t - \tau_{1,i} - l_1) \right) \left( \prod_{i=1} y_i(t - \tau_{3,i} - l_3) \right) \\ &= \left( \left( \sum_{\#\alpha} (\mathbf{G} \circledast \lceil \mathbf{H}_1, \alpha, \mathbf{H}_2 \rfloor) \right) * \lceil \vec{\mathbf{x}}, \vec{\mathbf{y}} \rfloor \right)(t).\end{aligned}$$

*Property 15.*

$$\begin{aligned}(\mathbf{h} * [\mathbf{x}]^n)(t) &= \sum_l h(l) \left(x(t-l)\right)^n \\ &= \sum_{\tau_1,\tau_2,\cdots,\tau_n} h(l) \prod_{i=1}^n \left(\delta(\tau_i - l) x(t - \tau_i)\right) \\ &= \sum_{\tau_1,\tau_2,\cdots,\tau_n} (\mathrm{diag}(n,\mathbf{h}))(\tau_1,\tau_2,\cdots,\tau_n) \prod_{i=1}^n x(t-\tau_i) \\ &= (\mathrm{diag}(n,\mathbf{h}) * \mathbf{x}^n)(t).\end{aligned}$$

*Property 16.*

$$\mathbf{g} * [\mathbf{h} * \mathbf{x}]^n = \mathrm{diag}(n,\mathbf{g}) * (\mathbf{h} * \mathbf{x})^n = (\mathrm{diag}(n,\mathbf{g}) \circledast \mathbf{h}^n) * \mathbf{x}^n.$$

■

## Appendix C. Perturbations in Volterra Convolution

During practice, it has been found that some specific perturbations have the potential to make neural networks behave abnormally (Szegedy et al., 2014; Goodfellow et al., 2015). An observation of how perturbations influence the proxy kernels will be theoretically explored in this appendix. The neural network is extremely complex, our analysis limits on its impacts on the proxy kernels.

### C.1 Perturbation Upper Bound

In this subsection, we express the upper bound for perturbation of ideal Volterra convolution, showing that how the perturbation change the output.

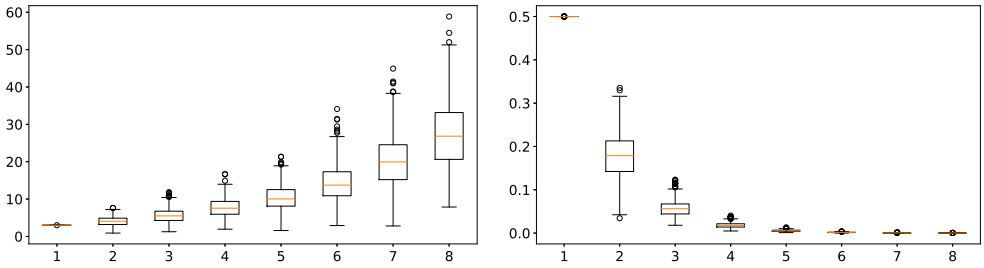
**Theorem 19** *Assume input signal is* $\mathbf{x}$*, and the perturbation is* $\epsilon$*, the approximated neural network is* $f(\mathbf{x}) = \sum_{n=0}^{N} \mathbf{H}_n * \mathbf{x}^n$*, we have*

$$\|f(\mathbf{x}+\epsilon) - f(\mathbf{x})\|_2 \leq \min \left( \begin{array}{l} \displaystyle\sum_{n=0}^{N} \|\mathbf{H}_n\|_2 \sum_{k=0}^{n-1} \left(\frac{en}{k}\right)^k \|\mathbf{x}\|_1^k \|\epsilon\|_1^{n-k}, \\ \displaystyle\sum_{n=0}^{N} \|\mathbf{H}_n\|_1 \sum_{k=0}^{n-1} \left(\frac{en}{k}\right)^k \|\mathbf{x}\|_{2k}^k \|\epsilon\|_{2(n-k)}^{n-k} \end{array} \right), \tag{34}$$

*where* $e = 2.718281828\cdots$*, the base of the natural logarithm.*

**Proof** In appendix C.3. ■

The following is an example of the effects of perturbations for each convolution from order one to eight. We randomly generate $\mathbf{x} \in \mathbb{M}^{32}$ ($\mathbb{M}$ has defined in Equation 27). Two perturbations are produced to simulate these two different cases. To make the plot more clear, we add a considerably large number 3.0 or a considerably small value 0.5 to the middle point of $\mathbf{x}$. For each perturbation and each convolution with order from one to eight, we compute $\|\mathbf{H}_n * (\mathbf{x}+\epsilon)^n - \mathbf{H}_n * \mathbf{x}^n\|_2$ one thousand times with random $\mathbf{H}_n \in \mathbb{M}^{5\times\cdots\times 5}$. The boxplot is shown in Figure 12.




(a) add 3.0 (b) add 0.5

Figure 12: Boxplot of $\|\mathbf{H}_n * (\mathbf{x}+\epsilon)^n - \mathbf{H}_n * \mathbf{x}^n\|_2, n = 1, 2, \cdots, 8$. The horizontal axis indicates the order and the vertical axis indicates the error.

Figure 12 shows that the bound is reasonable. It seems that the impact of perturbation will cause exponential blowup or decay. Nevertheless, this result is obtained under the context of ideal Volterra convolution. In reality, the situation will be more complicated, we need to consider the kernels, input signals, and so on.

If the impact decays, this means that the related neural network are robust to such perturbation, which is exactly what we want.

As for the exponential blowup, we need to consider the neural network before the approximation. If the neural network is Lipschitz in some domains, the approximated

Volterra convolution must be Lipschitz in the same domains. The upper bound of change of output is bounded by the Lipschitz constant, and the bound is not exponential. Otherwise, if the neural network is non-Lipschitz, but the output is bounded, this impact is also bounded. If the output of the neural network is not bounded, something wrong must have happened to this neural network. This is because all operations in a neural network is bounded, i.e., the convolutions are bounded if both kernel and signal are bounded, and the normalization operations are also bounded if not divide by zero. In conclusion, the impact of perturbation will increase but won't exponential blowup in practice.

### C.2 Perturbation on Order-n Volterra Convolution

The following is an example of how perturbations affect the order-$n$ Volterra convolution. Due to the computation complexity, $n$ is set to be eight,

$$f(\mathbf{x}) = \sum_{i=1}^{8} \mathbf{H}_i * \mathbf{x}^i, \mathbf{H}_0 = 0.$$

Kernels are randomly generated as $\mathbf{H}_1 \in \mathbb{M}^5, \mathbf{H}_2 \in \mathbb{M}^{5\times5}, \cdots \mathbf{H}_8 \in \mathbb{M}^{5\times\cdots\times5}$ ($\mathbb{M}$ is defined in Equation 27). Input signal is selected as a sin function $\mathbf{x} = \sin(t), 0 \leq t \leq 8\pi$. Thirty spots are picked from $\mathbf{x}$ and added 0.2 as perturbation $\mathbf{x} + \epsilon$. The perturbations are denser in the head and sparser in the tail. Result is illustrated in Figure 13.

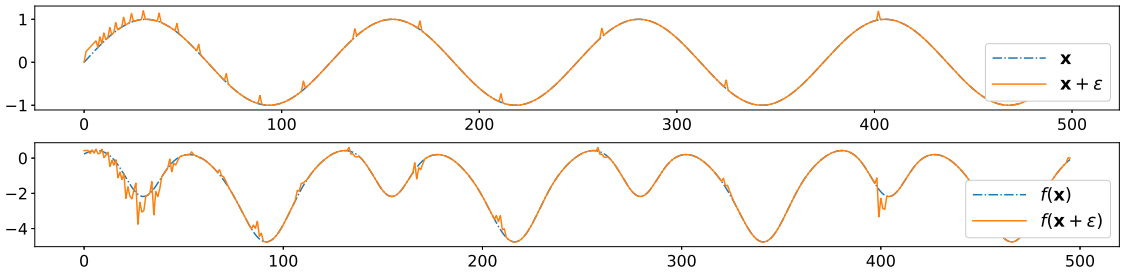



Figure 13: Example for perturbation of order-eight Volterra convolution.

This result is influenced by both parameters and perturbations. Different parameters will cause different results. Some are sensitive to perturbations while others are not.

As can be seen in Figure 13, regions in the head are more influenced by perturbations, while regions in the tail are not. This is because perturbations are denser in the head than in the tail, and short time energy is larger than that in the tail. It can also be seen that some special regions, such as mountain peaks or valleys, are more affected by the disturbance, while other regions are less affected. Not all perturbations are effective. The effective perturbations are determined by both kernels and input signals.

### C.3 Proof for Theorem 19

In the following, we prove Theorem 19. For proving this Lemma, we will first introduce Young's Inequality (Theorem 20), Corollary 21, Lemma 22 and Lemma 23.

**Theorem 20 (Young's Inequality for Convolutions (Henry, 1912))** *Let $p, q, r \in \mathbb{R}$, $p, q, r \geq 1$ and $1 + \frac{1}{r} = \frac{1}{p} + \frac{1}{q}$. For signals $\mathbf{h}$ and $\mathbf{x}$, following inequality is satisfied,*

$$\|\mathbf{h} * \mathbf{x}\|_r \leq \|\mathbf{h}\|_p \|\mathbf{x}\|_q. \tag{35}$$

**Corollary 21** *Let $r = p = 2, q = 1$. Theorem 20 is simplified as*

$$\|\mathbf{h} * \mathbf{x}\|_2 \leq \min\left(\|\mathbf{h}\|_2 \|\mathbf{x}\|_1, \|\mathbf{h}\|_1 \|\mathbf{x}\|_2\right). \tag{36}$$

**Lemma 22** *Suppose that $\mathbf{H}_n$ is the kernel, and $\mathbf{x}, \mathbf{y}$ are two signals. We have*

$$\left\| \mathbf{H}_n * \left\lceil \mathbf{x}^k, \mathbf{y}^{n-k} \right\rfloor \right\|_2 \leq \min \begin{pmatrix} \|\mathbf{H}_n\|_2 \|\mathbf{x}\|_1^n \|\mathbf{y}\|_1^{n-k}, \\ \|\mathbf{H}_n\|_1 \|\mathbf{x}\|_{2k}^k \|\mathbf{y}\|_{2(n-k)}^{n-k} \end{pmatrix}, \quad k = 0, 1, \cdots, n.$$

**Proof** By definition of order-$n$ convolution, we have

$$\left(\mathbf{H}_n * \left\lceil \mathbf{x}^k, \mathbf{y}^{n-k} \right\rfloor\right)(t) = \sum_{\tau_1, \cdots, \tau_n} H_n(\tau_1, \tau_2, \cdots, \tau_n) \prod_{i=1}^{k} x(t - \tau_i) \prod_{i=k+1}^{n} y(t - \tau_i) \tag{37}$$

Let $Z(t_1, t_2, \cdots, t_n) = \prod_{i=1}^{k} x(t_i) \prod_{i=k+1}^{n} y(t_i)$. Equation 37 becomes

$$\begin{aligned} \left(\mathbf{H}_n * \left\lceil \mathbf{x}^k, \mathbf{y}^{n-k} \right\rfloor\right)(t) &= \sum_{\tau_1, \cdots, \tau_n} H_n(\tau_1, \tau_2, \cdots, \tau_n) Z(t - \tau_1, t - \tau_2, \cdots, t - \tau_n) \\ &= (\mathbf{H}_n * \mathbf{Z})(t, t, \cdots, t). \end{aligned}$$

Recall Young's Inequality (Theorem 20),

$$\|\mathbf{H}_n * \mathbf{Z}\|_2 \leq \min\left(\|\mathbf{H}_n\|_2 \|\mathbf{Z}\|_1, \|\mathbf{H}_n\|_1 \|\mathbf{Z}\|_2\right),$$

where $\|\mathbf{Z}\|_1$ is

$$\begin{aligned} \|\mathbf{Z}\|_1 &= \sum_{\vec{\mathbf{t}}} |Z(t_1, t_2, \cdots, t_n)| \\ &= \sum_{\vec{\mathbf{t}}} \left| \prod_{i=1}^{k} x(t_i) \prod_{i=k+1}^{n} y(t_i) \right| \\ &= \left( \prod_{i=1}^{k} \sum_{t_1 \cdots} |x(t_i)| \right) \left( \prod_{i=k+1}^{n} \sum_{t_{k+1} \cdots} |y(t_i)| \right) \\ &= \|\mathbf{x}\|_1^k \|\mathbf{y}\|_1^{n-k}, \end{aligned}$$

and $\|\mathbf{Z}\|_2$ is

$$\begin{aligned}\|\mathbf{Z}\|_2 &= \sqrt{\sum_{t_1,t_2,\cdots,t_n}(Z(t_1,t_2,\cdots,t_n))^2} \\ &= \sqrt{\sum_{t_1,t_2,\cdots,t_n}\left(\prod_{i=1}^{k}x(t_i)\prod_{i=k+1}^{n}y(t_i)\right)^2} \\ &= \sqrt{\left(\sum_{t_1\cdots}x^{2k}(t_i)\right)\left(\sum_{t_{k+1}\cdots}y^{2(n-k)}(t_i)\right)} \\ &= \|\mathbf{x}\|_{2k}^{k}\|\mathbf{y}\|_{2(n-k)}^{n-k}.\end{aligned}$$

This proof is finished. ■

**Lemma 23** *Suppose that* $\mathbf{H}_n$ *is the kernel, and* $\mathbf{x},\mathbf{y}$ *are two signals. We have*

$$\|\mathbf{H}_n*(\mathbf{x}+\mathbf{y})^n-\mathbf{H}_n*\mathbf{x}^n\|_2 \le \min\begin{pmatrix}\|\mathbf{H}_n\|_2\sum_{k=0}^{n-1}\left(\frac{en}{k}\right)^k\|\mathbf{x}\|_1^k\|\mathbf{y}\|_1^{n-k}, \\ \|\mathbf{H}_n\|_1\sum_{k=0}^{n-1}\left(\frac{en}{k}\right)^k\|\mathbf{x}\|_{2k}^k\|\mathbf{y}\|_{2(n-k)}^{n-k}\end{pmatrix}, \tag{38}$$

*where* $k=0,1,\cdots,n.$, *and* $e=2.718281828\cdots$, *the base of the natural logarithm.*

**Proof** Recall Property 5,

$$\mathbf{H}_n*(\mathbf{x}+\mathbf{y})^n=\sum_{k=0}^{n}\binom{n}{k}\mathbf{H}_n*\left[\mathbf{x}^k,\mathbf{y}^{n-k}\right]. \tag{39}$$

We move $\mathbf{H}_n * \mathbf{x}^n$ to the left and take the $L_2$-norm. By Lemma 22, the $L_2$-norm is

$$\begin{aligned}
\|\mathbf{H}_n * (\mathbf{x}+\mathbf{y})^n - \mathbf{H}_n * \mathbf{x}^n\|_2 &= \left\| \sum_{k=0}^{n-1} \binom{n}{k} \mathbf{H}_n * \left\lceil \mathbf{x}^k, \mathbf{y}^{n-k} \right\rfloor \right\|_2 \\
&\leq \sum_{k=0}^{n-1} \binom{n}{k} \left\| \mathbf{H}_n * \left\lceil \mathbf{x}^k, \mathbf{y}^{n-k} \right\rfloor \right\|_2 \\
&\leq \min \begin{pmatrix} \|\mathbf{H}_n\|_2 \sum_{k=0}^{n-1} \binom{n}{k} \|\mathbf{x}\|_1^k \|\mathbf{y}\|_1^{n-k}, \\ \|\mathbf{H}_n\|_1 \sum_{k=0}^{n-1} \binom{n}{k} \|\mathbf{x}\|_{2k}^k \|\mathbf{y}\|_{2(n-k)}^{n-k} \end{pmatrix} \\
&\leq \min \begin{pmatrix} \|\mathbf{H}_n\|_2 \sum_{k=0}^{n-1} \left(\frac{en}{k}\right)^k \|\mathbf{x}\|_1^k \|\mathbf{y}\|_1^{n-k}, \\ \|\mathbf{H}_n\|_1 \sum_{k=0}^{n-1} \left(\frac{en}{k}\right)^k \|\mathbf{x}\|_{2k}^k \|\mathbf{y}\|_{2(n-k)}^{n-k} \end{pmatrix},
\end{aligned}$$

where binomial coefficient $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$ (Exercise 0.0.5 of Vershynin (2018)). ■

The following is the proof of Theorem 19.

**Proof** [Proof of Theorem 19]

$$\begin{aligned}
\|f(\mathbf{x}+\epsilon) - f(\mathbf{x})\|_2 &= \left\| \sum_{n=0}^{N} (\mathbf{H}_n * (\mathbf{x}+\epsilon)^n - \mathbf{H}_n * \mathbf{x}^n) \right\|_2 \\
&\leq \sum_{n=0}^{N} \|(\mathbf{H}_n * (\mathbf{x}+\epsilon)^n - \mathbf{H}_n * \mathbf{x}^n)\|_2 \qquad \text{(apply Lemma 23)} \\
&\leq \min \begin{pmatrix} \sum_{n=0}^{N} \|\mathbf{H}_n\|_2 \sum_{k=0}^{n-1} \left(\frac{en}{k}\right)^k \|\mathbf{x}\|_1^k \|\epsilon\|_1^{n-k}, \\ \sum_{n=0}^{N} \|\mathbf{H}_n\|_1 \sum_{k=0}^{n-1} \left(\frac{en}{k}\right)^k \|\mathbf{x}\|_{2k}^k \|\epsilon\|_{2(n-k)}^{n-k} \end{pmatrix}.
\end{aligned}$$

■

## Appendix D. Rank for Outer Convolution

In this appendix, we discuss the rank of the outer convolution in discrete time. The rank of a matrix equal to the number of non-zero eigenvalues. It is of major importance, which tells us linear dependencies of column or row vectors, and is one of the fundamental building block of matrix completion and compress sensing (Candès et al., 2006; Donoho, 2006; Sidiropoulos

et al., 2017). The rank of a tensor could be defined in various ways, such as the rank based on CANDECOMP/PARAFACT decomposition and Tucker decomposition and others (Kolda and Bader, 2009).

Before discussing the rank, let's first take a look at linear dependence and independence (Horn and Johnson, 1985). Vectors $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n$ are linear independence if $\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \cdots + \alpha_n\mathbf{v}_n = 0$ only implies all scalars are zero $\alpha_1 = \alpha_2 = \cdots = 0$. Similar to that, we call $\mathbf{G}_1, \mathbf{G}_2, \cdots, \mathbf{G}_n$ are linear independence if $\alpha_1\mathbf{G}_1 + \alpha_2\mathbf{G}_2 + \cdots + \alpha_n\mathbf{G}_n = 0$ only implies $\alpha_1 = \alpha_2 = \cdots = \alpha_n = 0$. If $k$ vectors span a space of dimension $r$, any of these $k$ vectors can be represented as a linear combination of $r$ linear independence vectors in that space.

To express the rank of matrix, Tucker rank (Sidiropoulos et al., 2017) of tensor, and others, we denote

$$\text{rank}(k, \mathbf{G}) = \begin{cases} 1, & \text{if } \mathbf{G} \text{ is 1 dimension} \\ \text{column rank of } \mathbf{G}, & \text{if } \mathbf{G} \text{ is 2 dimension, } k = 1 \\ \text{row rank of } \mathbf{G}, & \text{if } \mathbf{G} \text{ is 2 dimension, } k = 2 \\ k\text{-th Tucker rank of } \mathbf{G}, & \text{others} \end{cases}. \tag{40}$$

Especially, if $\mathbf{G}$ is a matrix or one-dimensional vector, this notation can be rewritten as $\text{rank}(\mathbf{G})$. Besides, a tensor $\mathbf{G}$ is non-zero means that its arbitrary Lp-norm $\|\mathbf{G}\|_p \neq 0, p \geq 1$.

Before talking about the rank, we would like to introduce the zero result of convolutions. By the definition of one-dimensional convolution, we can rewrite it into matrix multiplication format,

$$y(t) = \sum_{\tau=0}^{T} g(\tau)h(t-\tau) \Leftrightarrow \begin{bmatrix} \vdots \\ y(t) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ h(t-0) & h(t-1) & \cdots & h(t-T) \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} g(0) \\ g(1) \\ \vdots \\ g(T) \end{bmatrix},$$

where matrix generated by shifting $h(\cdot)$ is the Hankel matrix. It is clear that the convolution $\mathbf{g} * \mathbf{h}$ is zero if $\mathbf{g}$ is in the null space (Horn and Johnson, 1985) of the Hankel matrix. For example, the zero padding one-dimensional convolution is zero if kernel and signal have the following format,

$$\begin{aligned} \mathbf{g} &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}; \\ \mathbf{h} &= \begin{bmatrix} 1 & -1 & 0 & -1 & 1 & 1 & -1 & 0 & -1 & \cdots \end{bmatrix}. \end{aligned}$$

The high-dimensional convolution also follows the same rule. We can rewrite this kind of convolutions into matrix multiplication format by flattening $\mathbf{G}$ and the patches of $\mathbf{H}$. A patch of $\mathbf{H}$ is $H(\vec{\mathbf{t}} - \lceil \tau_1, \tau_2, \cdots, \tau_k \rfloor)$, where $\tau_i = 0, 1, \cdots, N_i$ with $i = 1, 2, \cdots, k$ and $\mathbf{G} \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_k}$. For instance, a two-dimensional convolution can be rewritten into the following format,

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ H(t_1-0, t_2-0) & H(t_1-0, t_2-1) & H(t_1-1, t_2-0) & H(t_1-1, t_2-1) \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} G(0,0) \\ G(0,1) \\ G(1,0) \\ G(1,1) \end{bmatrix}.$$

If the flatted $\mathbf{G}$ is in the null space of the matrix generated by flattening patches of $\mathbf{H}$, the result of this high-dimensional convolution is zero.

### D.1 Linear Independence in Convolutions

**Lemma 24** *There are $r \leq \min(\mathit{rank}(\mathcal{H}), \mathit{rank}(\mathcal{G}))$ linear independence tensors in the linear combinations of $\mathbf{G}_i * \mathbf{H}, i = 1, \cdots$, where $\mathcal{H}$ is the matrix generated by flattening patches of $\mathbf{H}$, and $\mathcal{G}$ is the matrix generated by flattening $\mathbf{G}_1, \mathbf{G}_2, \cdots$. If $\mathcal{H}$ is full column rank, $r = \mathit{rank}(\mathcal{G})$.*

**Proof**

Suppose $Y_i(\vec{\mathbf{t}}) = \sum_{\vec{\tau}} H(\vec{\mathbf{t}} - \vec{\tau}) G_i(\vec{\tau})$, we rewrite this convolution into matrix multiplication format,

$$\begin{bmatrix} | & | & \\ Y_1(\vec{\mathbf{t}}) & Y_2(\vec{\mathbf{t}}) & \cdots \\ | & | & \end{bmatrix} = \begin{bmatrix} & \vdots & \\ \cdots & H(\vec{\mathbf{t}} - \vec{\tau}) & \cdots \\ & \vdots & \end{bmatrix} \begin{bmatrix} | & | & \\ G_1(\vec{\tau}) & G_2(\vec{\tau}) & \cdots \\ | & | & \end{bmatrix} = \mathcal{H}\mathcal{G}.$$

Let matrix $\mathcal{Y}$ be the matrix generated by flattening $\mathbf{Y}_1, \mathbf{Y}_2, \cdots$. We have (Horn and Johnson, 1985)

$$r = \operatorname{rank}(\mathcal{Y}) = \operatorname{rank}(\mathcal{H}\mathcal{G}) \leq \min\left(\operatorname{rank}(\mathcal{H}), \operatorname{rank}(\mathcal{G})\right).$$

If $\mathcal{H}$ is not full column rank, it is possible that some linear combinations of the columns of $\mathcal{G}$ are in the null space of $\mathcal{H}$. The matrix multiplication with $\mathcal{H}$ and these combinations are zero, which implies that the convolution of $\mathbf{H}$ and the same combinations of $\mathbf{G}_i$ are zero.

Otherwise, if $\mathcal{H}$ is full column rank, left multiplication by a full column rank matrix leaves rank unchanged, $r = \operatorname{rank}(\mathcal{H}\mathcal{G}) = \operatorname{rank}(\mathcal{G})$, and $\operatorname{rank}(\mathcal{G})$ equals the number of linear independence tensors in the linear combinations of $\mathbf{G}_1, \mathbf{G}_2, \cdots$. It means that there are no combinations of $\mathbf{G}_i$ that lead to a zero convolution result.

■

By Lemma 24, if the $\mathcal{H}$ is full column rank, the number of linear independent tensors in linear combinations of $\mathbf{G}_1 * \mathbf{H}, \mathbf{G}_2 * \mathbf{H}, \cdots$ equals to that of $\mathbf{G}_1, \mathbf{G}_2, \cdots$. For instance, if $\mathbf{H}$ meets the condition, and $\mathbf{G}_1 \neq \alpha \mathbf{G}_2$ for any scalar $\alpha$, we have $\mathbf{G}_1 * \mathbf{H} \neq \alpha \mathbf{G}_2 * \mathbf{H}$.

### D.2 Matrix Rank for Two-dimensional Signals

**Lemma 25** *$\mathit{rank}(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor) \leq \min(\mathit{rank}(\mathbf{G}), \mathit{rank}(\mathcal{H}_1), \mathit{rank}(\mathcal{H}_2)) \leq \mathit{rank}(\mathbf{G})$, where $\mathbf{G} \in \mathbb{R}^{n_1 \times n_2}$, and $\mathbf{h}_1 \in \mathbb{R}^{l_1}, \mathbf{h}_2 \in \mathbb{R}^{l_2}$. We get equality, $\mathit{rank}(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor) = \mathit{rank}(\mathbf{G})$, if $\mathcal{H}_1 \in \mathbb{R}^{(l_1 + 2p_1 - n_1) \times n_1}$ and $\mathcal{H}_2 \in \mathbb{R}^{(l_2 + 2p_2 - n_2) \times n_2}$, the two matrices generated by shifting $h_1(\cdot)$ and $h_2(\cdot)$ with padding size $p_1$ and $p_2$, are full column rank.*

**Proof** Since the rank of $\mathbf{G}$ is $\operatorname{rank}(\mathbf{G})$, we can factorize $\mathbf{G}$ into the form of summation by outer product of linear independent vectors,

$$G(\tau_1, \tau_2) = \sum_{r=1}^{\operatorname{rank}(\mathbf{G})} p_r(\tau_1) q_r(\tau_2).$$

By the definition of outer convolution (Definition 12), we have

$$\begin{aligned}(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor)(t_1, t_2) &= \sum_{\tau_1, \tau_2} G(\tau_1, \tau_2) h_1(t_1 - \tau_1) h_2(t_2 - \tau_2) \\ &= \sum_{r=1}^{\text{rank}(\mathbf{G})} \left( \sum_{\tau_1} p_r(\tau_1) h_1(t_1 - \tau_1) \right) \left( \sum_{\tau_2} q_r(\tau_2) h_2(t_2 - \tau_2) \right). \\ &= \sum_{r=1}^{\text{rank}(\mathbf{G})} (\mathbf{p}_r * \mathbf{h}_1)(t_1)(\mathbf{q}_r * \mathbf{h}_2)(t_2),\end{aligned}$$

which is a summation of the outer product of vectors $\mathbf{p}_r * \mathbf{h}_1$ and $\mathbf{q}_r * \mathbf{h}_2$.

Recall Lemma 24, we have

$$\begin{aligned}\text{rank}(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor) &\leq \min\left(\min(\text{rank}(\mathbf{G}), \text{rank}(\mathcal{H}_1)), \min(\text{rank}(\mathbf{G}), \text{rank}(\mathcal{H}_2))\right) \\ &\leq \min\left(\text{rank}(\mathbf{G}), \text{rank}(\mathcal{H}_1), \text{rank}(\mathcal{H}_2)\right) \\ &\leq \text{rank}(\mathbf{G}).\end{aligned}$$

If $\mathcal{H}_1$ is full column rank, there is no such vector $\mathbf{v} \in \mathbb{R}^{n_1}$ such that $\mathbf{v} * \mathbf{h}_1$ is zero, and $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_{\text{rank}(\mathbf{G})}$ are linear independence. If $\mathcal{H}_2$ is full column rank, the same rule can also be applied to $\mathcal{H}_2$. Under this assumption, we have

$$\text{rank}(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor) = \text{rank}(\mathbf{G}).$$

■

Let $H(t_1, t_2) = h_1(t_1) h_2(t_2)$. It becomes a two-dimensional convolution,

$$\begin{aligned}\left(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor\right)(t_1, t_2) &= \sum_{\tau_1, \tau_2} G(\tau_1, \tau_2) h_1(t_1 - \tau_1) h_2(t_2 - \tau_2) \\ &= \sum_{\tau_1, \tau_2} G(\tau_1, \tau_2) H(t_1 - \tau_1, t_2 - \tau_2).\end{aligned}$$

More generally, we extend this to the $\mathbf{H}$ of rank grater than or equal to one.

**Lemma 26** *rank*$(\mathbf{G} * \mathbf{H}) \leq \min(s_1, s_2, \textit{rank}(\mathbf{G})\textit{rank}(\mathbf{H}))$*, where both* $\mathbf{G}$ *and* $\mathbf{H}$ *are two-dimensional matrices, and* $\mathbf{G} * \mathbf{H} \in \mathbb{R}^{s_1 \times s_2}$*.*

**Proof**

Suppose that $\widehat{\mathbf{H}}_1, \widehat{\mathbf{H}}_2, \cdots, \widehat{\mathbf{H}}_{\text{rank}(\mathbf{H})}$ are linear independence matrices, and $\mathbf{H} = \sum_{r=1}^{\text{rank}(\mathbf{H})} \widehat{\mathbf{H}}_r$. The convolution becomes

$$\mathbf{G} * \mathbf{H} = \mathbf{G} * \left( \sum_{r=1}^{\text{rank}(\mathbf{H})} \widehat{\mathbf{H}}_r \right) = \sum_{r=1}^{\text{rank}(\mathbf{H})} \mathbf{G} * \widehat{\mathbf{H}}_r.$$

By Lemma 25, the rank of this convolution is

$$\text{rank}\left(\mathbf{G} * \mathbf{H}\right) = \text{rank}\left( \sum_{r=1}^{\text{rank}(\mathbf{H})} \mathbf{G} * \widehat{\mathbf{H}}_r \right) \leq \sum_{r=1}^{\text{rank}(\mathbf{H})} \text{rank}(\mathbf{G}) \leq \text{rank}(\mathbf{H})\text{rank}(\mathbf{G}).$$

In addition, the rank of a matrix must be bounded by its size, implying that

$$\operatorname{rank}(\mathbf{G} * \mathbf{H}) \leq \min(s_1, s_2, \operatorname{rank}(\mathbf{G})\operatorname{rank}(\mathbf{H})).$$

■

**Remark 27** *The two-dimensional convolution often increase the rank of a two-dimensional image.*

**Remark 28** *Dilated convolutions (Yu and Koltun, 2016) still follow Lemma 26, since adding zero-filled rows or columns will not alter the rank.*

### D.3 Tucker Rank for Multi-dimensional Signals

**Lemma 29** *rank*$(k, \mathbf{G} \circledast \vec{\mathbf{h}}) \leq$ *rank*$(k, \mathbf{G})$*, where* $k = 1, 2, \cdots, n$*, and* $\vec{\mathbf{h}} = \lceil \mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n \rfloor$ *is a list of non-zero one-dimensional signals, and* $\mathbf{G}$ *is an* $n$*-dimensional tensor, and rank*$(k, \mathbf{G})$ *is the* $k$*-th Tucker rank of* $\mathbf{G}$*.*

**Proof** If $n = 2$, this has been proved in Lemma 25.

If $n > 2$, suppose that $\mathbf{G} \in \mathbb{R}^{s_1 \times s_2 \times \cdots \times s_n}$. Focusing on the $k$-th dimension, we have

$$\begin{aligned}(\mathbf{G} \circledast \vec{\mathbf{h}})(\vec{\mathbf{t}}) &= \sum_{\vec{\tau}} G(\vec{\tau}) \prod_{i=1}^{n} h_i(t_i - \tau_i) \\ &= \sum_{\tau_k} \left( \sum_{\tau_1, \cdots, \tau_{k-1}} \sum_{\tau_{k+1}, \cdots, \tau_n} G(\vec{\tau}) \prod_{i=1, i \neq k}^{n} h_i(t_i - \tau_i) \right) h_k(t_k - \tau_k).\end{aligned}$$

Let

$$P(t_1, \cdots, t_{k-1}, \tau_k, t_{k+1}, \cdots, t_n) = \sum_{\tau_1, \cdots, \tau_{k-1}} \sum_{\tau_{k+1}, \cdots, \tau_n} G(\vec{\tau}) \prod_{i=1, i \neq k}^{n} h_i(t_i - \tau_i).$$

The convolution becomes

$$(\mathbf{G} \circledast \vec{\mathbf{h}})(\cdots, t_k, \cdots) = \sum_{\tau_k} P(\cdots, \tau_k, \cdots) h_k(t_k - \tau_k).$$

We permute and reshape $\mathbf{P}$ into matrix format $\widehat{\mathbf{P}} \in \mathbb{R}^{s_k \times \cdot}$, and rewrite the convolution into matrix multiplication format,

$$\begin{bmatrix} \vdots & \vdots & & \vdots \\ h_k(t_k - 0) & h_k(t_k - 1) & \cdots & h_k(t_k - s_k) \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} \hat{P}(0,1) & \hat{P}(0,2) & \cdots \\ \hat{P}(1,1) & \hat{P}(1,2) & \cdots \\ \vdots & \vdots & \\ \hat{P}(s_k,1) & \hat{P}(s_k,2) & \cdots \end{bmatrix} \equiv \mathcal{H}_k \widehat{\mathbf{P}}.$$

By the rank inequality of matrix multiplication, we have

$$\begin{aligned}\operatorname{rank}(k, \mathbf{G} \circledast \vec{\mathbf{h}}) &\leq \min(\operatorname{rank}(\mathcal{H}_k), \operatorname{rank}(\widehat{\mathbf{P}})) \\ &\leq \min(\operatorname{rank}(\mathcal{H}_k), \operatorname{rank}(k, \mathbf{G})) \quad \text{(Lemma 24)} \\ &\leq \operatorname{rank}(k, \mathbf{G}).\end{aligned}$$

■

Let $H(t_1, t_2, \cdots, t_n) = h_1(t_1)h_2(t_2)\cdots h_n(t_n)$, which is a tensor with Tucker rank equals $[1, 1, \cdots, 1]$. We have $\mathbf{G} \circledast \vec{\mathbf{h}} = \mathbf{G} * \mathbf{H}$, and we extend this to a more general $\mathbf{H}$ in Lemma 30.

**Lemma 30** *rank*$(k, \mathbf{G} * \mathbf{H}) \leq \min(s_k,$ *rank*$(k, \mathbf{G})$*rank*$(k, \mathbf{H}))$*, where* $\mathbf{G}, \mathbf{H}$ *are both non-zero* $n$*-dimensional tensors, and* $\mathbf{G} * \mathbf{H} \in \mathbb{R}^{s_1 \times s_2 \times \cdots \times s_n}$*.*

**Proof** If $n = 2$, it has been proved in Lemma 26.

If $n > 2$, suppose that

$$\mathbf{H} = \sum_{r=1}^{\text{rank}(k,\mathbf{H})} \widehat{\mathbf{H}}_r,$$

where $\mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_{\text{rank}(k,\mathbf{H})}$ are linear independent and all $\text{rank}(k, \widehat{\mathbf{H}}_r) = 1$. The $n$-dimensional convolution becomes,

$$\mathbf{G} * \mathbf{H} = \mathbf{G} * \left( \sum_{r=1}^{\text{rank}(k,\mathbf{H})} \widehat{\mathbf{H}}_r \right) = \sum_{r=1}^{\text{rank}(k,\mathbf{H})} \mathbf{G} * \widehat{\mathbf{H}}_r.$$

By Lemma 30, the $k$-th Tucker rank of this convolution is

$$\begin{aligned} \text{rank}\,(k, \mathbf{G} * \mathbf{H}) &= \text{rank}\left( k, \sum_{r=1}^{\text{rank}(k,\mathbf{H})} \mathbf{G} * \widehat{\mathbf{H}}_r \right) \\ &\leq \sum_{r=1}^{\text{rank}(k,\mathbf{H})} \text{rank}\,(k, \mathbf{G}) \\ &\leq \text{rank}(k, \mathbf{H})\text{rank}(k, \mathbf{G}). \end{aligned}$$

In addition, the $k$-th Tucker rank is bounded by the size, which implies that

$$\text{rank}(k, \mathbf{G} * \mathbf{H}) \leq \min(s_k, \text{rank}(k, \mathbf{G})\text{rank}(k, \mathbf{H})).$$

■

### D.4 Main Theorem of Rank for Outer Convolution

Now, all previous discussion of outer convolutions are combined, expanding Lemma 29 to Theorem 31. To simplify discussions, outer convolution is divided into groups,

$$(\mathbf{G} \circledast \vec{\mathbf{H}}) \left( \underbrace{((1,1),(1,2),\cdots)}_{\text{group 1, } (\mathbf{H}_1)}, \cdots, \underbrace{((i,1),(i,2),\cdots)}_{\text{group } k, \ (\mathbf{H}_k)} \right).$$

**Theorem 31 (Rank for outer convolution)**

$$rank(\lceil k,i \rfloor, \mathbf{G} \circledast \vec{\mathbf{H}}) \leq \begin{cases} \min\left(s_{k,1}, rank(k,\mathbf{G})\right), & \mathbf{H}_k \text{ is one-dimensional} \\ \min\left(s_{k,i}, z_k rank(i,\mathbf{H}_k)\right), & otherwise \end{cases},$$

*where* $\vec{\mathbf{H}} = \lceil \mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_n \rfloor$ *is a list of tensors, and* $\mathbf{G} \in \mathbb{R}^{z_1 \times z_2 \times \cdots \times z_n}$*, and* $\mathbf{G} \circledast \vec{\mathbf{H}} \in \mathbb{R}^{s_{1,1} \times s_{1,2} \times \cdots \times s_{2,1} \times s_{2,2} \times \cdots}$.

**Proof**

If $\mathbf{H}_k$ is one-dimensional, this proof is similar to that of Lemma 29. The $k$-th Tucker rank is

$$\text{rank}(\lceil k,1 \rfloor, \mathbf{G} \circledast \vec{\mathbf{H}}) \leq \min\left(s_{k,1}, \text{rank}(k,\mathbf{G})\right).$$

If $\mathbf{H}_k$ is not one-dimensional, we focus on the operations in group $k$. By the definition of outer convolution (Equation 13), we have

$$\begin{aligned}(\mathbf{G} \circledast \vec{\mathbf{H}})(\vec{\mathbf{t}}_1, \vec{\mathbf{t}}_2, \cdots, \vec{\mathbf{t}}_n) &= \sum_{\vec{\tau}} G(\vec{\tau}) \prod_{i=1}^{n} H_i(\vec{\mathbf{t}}_i - \tau_i) \\ &= \sum_{\tau_k} \left( \sum_{\tau_1,\cdots,\tau_{k-1}} \sum_{\tau_{k+1},\cdots,\tau_n} G(\vec{\tau}) \prod_{i=1, i\neq k}^{n} H_i(\vec{\mathbf{t}}_i - \tau_i) \right) H_k(\vec{\mathbf{t}}_k - \tau_k).\end{aligned}$$

Let

$$P(\vec{\mathbf{t}}_1, \cdots, \vec{\mathbf{t}}_{k-1}, \tau_k, \vec{\mathbf{t}}_{k+1}, \cdots, \vec{\mathbf{t}}_n) = \sum_{\tau_1,\cdots,\tau_{k-1}} \sum_{\tau_{k+1},\cdots,\tau_n} G(\vec{\tau}) \prod_{i=1, i\neq k}^{n} H_i(\vec{\mathbf{t}}_i - \tau_i).$$

The convolution becomes

$$\begin{aligned}(\mathbf{G} \circledast \vec{\mathbf{H}})(\cdots, \vec{\mathbf{t}}_k, \cdots) &= \sum_{\tau_k} P(\cdots, \tau_k, \cdots) H_k(\vec{\mathbf{t}}_k - \tau_k) \\ &= \sum_{\iota_1, \iota_2, \cdots} \hat{P}(\cdots; \iota_1, \iota_2, \cdots; \cdots) H_k(t_{k,1} - \iota_1, t_{k,2} - \iota_2, \cdots),\end{aligned}$$

where $\widehat{\mathbf{P}}$ is a super-diagonal format of $\mathbf{G}$,

$$\hat{P}(\cdots; \iota_1, \iota_2, \cdots; \cdots) = \begin{cases} P(\cdots, \tau_k, \cdots), & \tau_k = \iota_1 = \iota_2 = \cdots \\ 0, & \text{otherwise} \end{cases}.$$

This operation can be considered as a collection of multidimensional convolutions in group $k$, where all kernels are super-diagonal tensors. The Tucker rank of these super-diagonal tensors are $\text{rank}\left(\lceil k,i \rfloor, \widehat{\mathbf{P}}\right) \leq z_k$, with equality if and only if all diagonal entries are non-zero. Recall Lemma 30, we conclude that Tucker rank of this group is

$$\begin{aligned}\text{rank}\left(\lceil k,i \rfloor, \mathbf{G} \circledast \vec{\mathbf{H}}\right) &\leq \min(s_{k,i}, \text{rank}\left(\lceil k,i \rfloor, \widehat{\mathbf{P}}\right) \text{rank}(i,\mathbf{H}_k)) \\ &\leq \min(s_{k,i}, z_k \text{rank}(i,\mathbf{H}_k)).\end{aligned}$$

In conclusion, we complete this proof. ■

### D.5 Validation of the Rank Properties

*Data description.* To cover more situations, the following tensors are taken from three sets, $\mathbb{M}$, $\mathbb{U}$ and $\mathbb{O}$.

The first set is the Gaussian distribution with normalization,

$$\mathbb{M} = \left\{ \mathbf{x} : \mathbf{x} = \frac{\mathbf{y}}{\|\mathbf{y}\|_2}, y(\cdot) \sim \mathcal{N}(0,1) \right\},$$

where $\mathcal{N}(0,1)$ is standard Gaussian distribution with mean zero and variance one. This is the same as Equation 27.

The second set is the uniform distribution with normalization,

$$\mathbb{U} = \left\{ \mathbf{x} : \mathbf{x} = \frac{\mathbf{y}}{\|\mathbf{y}\|_2}, y(\cdot) \sim \mathcal{U}(0,1) \right\},$$

where $\mathcal{U}(0,1)$ is the uniform distribution from zero to one.

The third set is

$$\mathbb{O} = \{ \mathbf{h} : \exists \mathbf{g},\ \mathbf{g} * \mathbf{h} \text{ is zero}, \mathbf{g} \text{ and } \mathbf{h} \text{ are non-zero} \}.$$

One method for generating the desired signal $\mathbf{h}$ with a known kernel $\mathbf{g}$ is illustrated as follows. For any fixed index $t$, if $(\mathbf{g} * \mathbf{h})(t) = 0$, we have

$$\sum_{\tau=0}^{T} g(\tau) h(t-\tau) = g(0)h(t) + \sum_{\tau=1}^{T} g(\tau) h(t-\tau) = 0.$$

If the weighted sum $\sum_{\tau=1}^{T} g(\tau)h(t-\tau)$ is known, and $g(0)$ is non-zero, $h(t)$ can be uniquely determined by

$$h(t) = \frac{-1}{g(0)} \sum_{\tau=1}^{T} g(\tau) h(t-\tau). \tag{41}$$

We can iteratively compute the upcoming sequence $h(T), h(T+1), \cdots$, if all initial terms, $h(0), h(1), \cdots, h(T-1)$, are manually specified. It is clear that we cannot pad any other elements to the head or tail of $\mathbf{h}$, because the new elements may not follow Equation 41, and padding these elements may leave $\mathbf{g} * \mathbf{h}$ non-zero.

Let the matrix generated by shifting $\mathbf{h}$ be

$$\mathcal{H} = \left[\begin{array}{cccc|cc} h(0) & h(1) & \cdots & h(T-1) & h(T+0) & \cdots \\ h(1) & h(2) & \cdots & h(T+0) & h(T+1) & \cdots \\ h(2) & h(3) & \cdots & h(T+1) & h(T+2) & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \end{array}\right].$$

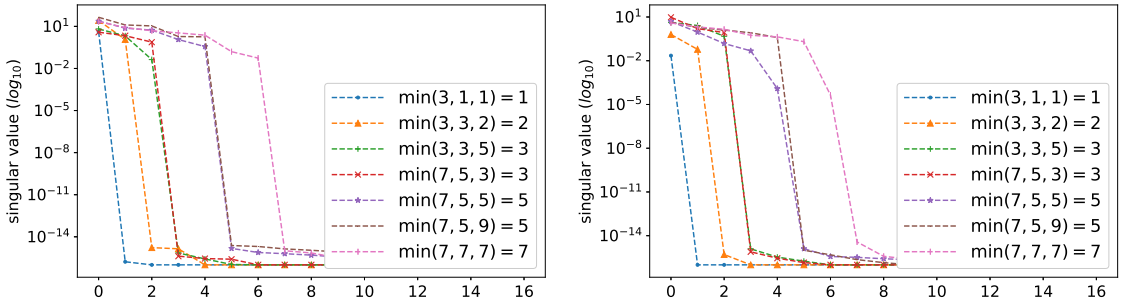
According to the computing process, the columns in the right of vertical line are linear combinations of the columns in the left. It implies that $\text{rank}(\mathcal{H}) \leq T$, with equality if the columns in the left of vertical line are linear independence.

*Validation process.* The main object is to compare the numerical rank with the theoretical rank. To begin with, we compute the convolutions of the generated kernels and signals. After

that, we clip the singular values of the convolution output to range $[1e^{-16}, 1e^{16}]$. At last, we plot the singular values to a figure with y-axis scaled by $\log_{10}$. We can easily distinguish the zero singular values from the non-zero ones in the figures. If there is a sharp slope, the singular values in the left of this slop are non-zero, and the number of the non-zero singular values equals the rank.

*Validate Lemma 25.* $\text{rank}(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor) \leq \min(\text{rank}(\mathbf{G}), \text{rank}(\mathcal{H}_1), \text{rank}(\mathcal{H}_2))$.

$\mathbf{G}$ is randomly generated with $\text{rank}(\mathbf{G}) = r_g$. $\mathbf{h}_1$ is iteratively computed by $T_1$ random initial terms and kernel $[1, 1, \cdots, 1]$. $\mathbf{h}_2$ is computed in the same way. Since zero padding of $\mathbf{h}_1$ or $\mathbf{h}_2$ is not allowed here, the outer convolution $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor$ is zero padded. The singular values are plotted in Figure 14. As shown in figure, $\text{rank}(\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor)$ is less than the smallest number of $r_g, T_1, T_2$, and $r_g = \text{rank}(\mathbf{G}), \text{rank}(\mathcal{H}_1) \leq T_1, \text{rank}(\mathcal{H}_2) \leq T_2$.

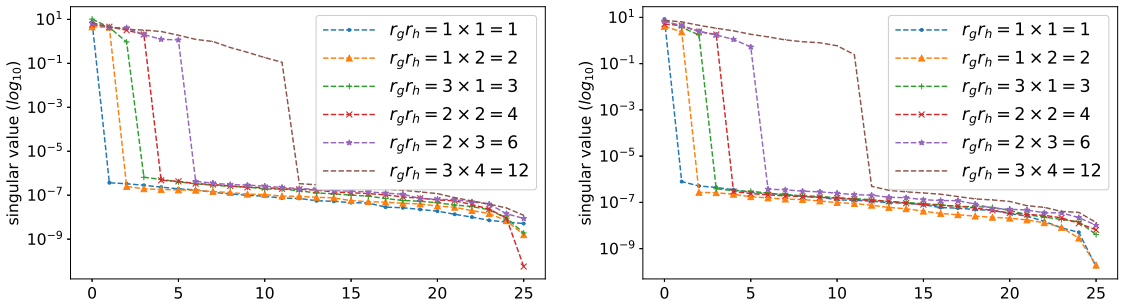



(a) $\mathbf{G} \in \mathbb{M}^{9\times 9}$, $\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{O}^{27}$ (b) $\mathbf{G} \in \mathbb{U}^{9\times 9}$, $\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{O}^{27}$

Figure 14: The singular values of $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2 \rfloor$ with no padding. The min operator with three numbers is $\min(r_g, T_1, T_2)$.

*Validate Lemma 26.* Regardless of size, the rank of a two-dimensional convolution $\mathbf{G} * \mathbf{H}$ is $\text{rank}(\mathbf{G} * \mathbf{H}) \leq \text{rank}(\mathbf{G})\text{rank}(\mathbf{H})$.

Two matrices $\mathbf{G}$ and $\mathbf{H}$ are randomly generated with $\text{rank}(\mathbf{G}) = r_g$ and $\text{rank}(\mathbf{H}) = r_h$. The singular values of $\mathbf{G} * \mathbf{H}$ are plotted in Figure 15. As the figure shows, $\text{rank}(\mathbf{G} * \mathbf{H})$ is less than or equal to the multiplication of $\text{rank}(\mathbf{G})$ and $\text{rank}(\mathbf{H})$.

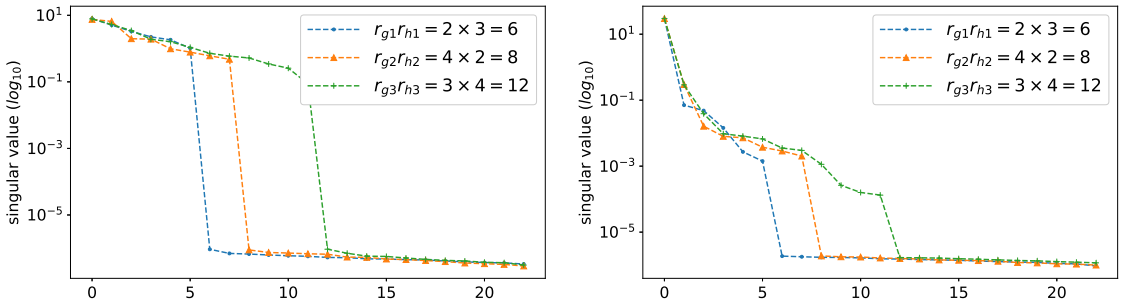



(a) $\mathbf{G} \in \mathbb{M}^{7\times 7}$, $\mathbf{H} \in \mathbb{M}^{32\times 32}$ (b) $\mathbf{G} \in \mathbb{U}^{7\times 7}$, $\mathbf{H} \in \mathbb{U}^{32\times 32}$

Figure 15: Singular values of two-dimensional convolutions $\mathbf{G} * \mathbf{H}$.

*Validate Lemma 30.* Regardless of size, the Tucker rank of an $n$-dimensional convolution is $\text{rank}(k, \mathbf{G} * \mathbf{H}) \leq \text{rank}(k, \mathbf{G})\text{rank}(k, \mathbf{H})$. This inequality of three-dimensional convolution is validated below.

$\mathbf{G}$ is randomly generated with Tucker rank equals $[2, 4, 3]$, and $\mathbf{H}$ is randomly generated with Tucker rank equals $[3, 2, 4]$. The $k$-th Tucker rank of a tensor equals to the matrix rank of mode-$k$ matricization of that tensor, where the mode-$k$ matricization is to permute and reshape the tensor with shape $(\cdots, s, \cdots)$ to $(\cdots, s)$ (Kolda and Bader, 2009). The singular values of mode-(one, two, three) matricization of $\mathbf{G} * \mathbf{H}$ are plotted in Figure 16. The $k$-th Tucker rank of $\mathbf{G} * \mathbf{H}$ is less than or equal to the multiplication of the $k$-th Tucker rank of $\mathbf{G}$ and $\mathbf{H}$.




(a) $\mathbf{G} \in \mathbb{M}^{6\times6\times6}$, $\mathbf{H} \in \mathbb{M}^{28\times28\times28}$ (b) $\mathbf{G} \in \mathbb{U}^{6\times6\times6}$, $\mathbf{H} \in \mathbb{U}^{28\times28\times28}$

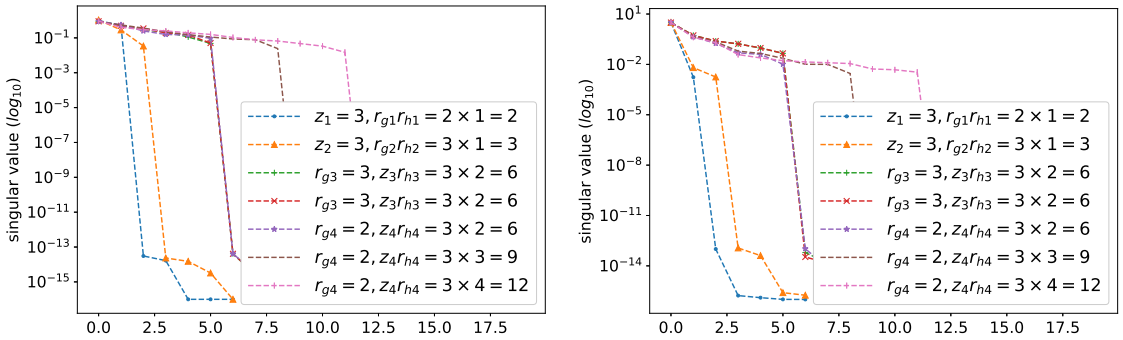
Figure 16: Singular values of mode-(one, two, three) matricization of three-dimensional convolution $\mathbf{G} * \mathbf{H}$.

*Validate Theorem 31 and Lemma 29.* Since Lemma 29 is a special case of Theorem 31 with all signals are one-dimensional, Theorem 31 is mainly validate below. Regardless of size, The Tucker rank of outer convolution is

$$\text{rank}(\lceil k, i \rfloor, \mathbf{G} \circledast \vec{\mathbf{H}}) \leq \begin{cases} \text{rank}(k, \mathbf{G}), & \mathbf{H}_k \text{ is one-dimensional} \\ z_k \text{rank}(i, \mathbf{H}_k), & \text{otherwise} \end{cases},$$

where $\vec{\mathbf{H}} = \lceil \mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_n \rfloor$ is a list of tensors, and $\mathbf{G} \in \mathbb{R}^{z_1 \times z_2 \times \cdots \times z_n}$. This inequality of outer convolution $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2, \mathbf{H}_3, \mathbf{H}_4 \rfloor$ is validated below.

Four-dimensional tensor $\mathbf{G}$ is randomly generated with Tucker rank equals $[2, 3, 3, 2]$. $\mathbf{h}_1$ and $\mathbf{h}_2$ are one-dimensional vectors, $\mathbf{H}_3$ is a two-dimensional matrix with $\text{rank}(\mathbf{H}_3) = 2$, and $\mathbf{H}_4$ is a three-dimensional tensor with Tucker rank equals $[2, 3, 4]$. The singular values of mode-$k$ matricization of $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2, \mathbf{H}_3, \mathbf{H}_4 \rfloor$ is plotted in Figure 17. $\mathbf{h}_1$ and $\mathbf{h}_2$ are one-dimensional, the first and second Tucker rank of $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2, \mathbf{H}_3, \mathbf{H}_4 \rfloor$ is less than or equal to that of $\mathbf{G}$. $\mathbf{H}_3$ and $\mathbf{H}_4$ are not one-dimensional, the corresponding Tucker rank of $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2, \mathbf{H}_3, \mathbf{H}_4 \rfloor$ is not grater than $z_k$ times the Tucker rank of $\mathbf{H}_3$ and $\mathbf{H}_4$.




(a) $\mathbf{G} \in \mathbb{M}^{3\times3\times3\times3}$, $\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{M}^5$, $\mathbf{H}_3 \in \mathbb{M}^{7\times7}$, $\mathbf{H}_4 \in \mathbb{M}^{9\times9\times18}$

(b) $\mathbf{G} \in \mathbb{U}^{3\times3\times3\times3}$, $\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{U}^5$, $\mathbf{H}_3 \in \mathbb{U}^{7\times7}$, $\mathbf{H}_4 \in \mathbb{U}^{9\times9\times18}$

Figure 17: Singular values of mode-$k$, $k = 1, 2, \cdots, 7$, matricization of $\mathbf{G} \circledast \lceil \mathbf{h}_1, \mathbf{h}_2, \mathbf{H}_3, \mathbf{H}_4 \rfloor$.

## References


Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. 2016. URL http://arxiv.org/abs/1607.06450.

Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information Theory, 39(3):930–945, may 1993. ISSN 00189448. doi: 10.1109/18.256500.

Stephen Boyd and Leon O. Chua. Fading Memory and the Problem of Approximating Nonlinear Operators With Volterra Series. IEEE transactions on circuits and systems, CAS-32(11):1150–1161, 1985. ISSN 00984094. doi: 10.1109/tcs.1985.1085649.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. 2021. URL http://arxiv.org/abs/2104.13478.

Richard A. Brualdi. Introductory combinatorics. Pearson/Prentice Hall, 4 edition, 2004.

Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust Uncertainty Principles : Exact Signal Frequency Information. IEEE Transactions on Information Theory, 52(2): 489–509, 2006.

G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems, 2(4):303–314, dec 1989. ISSN 0932-4194. doi: 10.1007/BF02551274. URL http://link.springer.com/10.1007/BF02551274.

David L. Donoho. For most large underdetermined systems of linear equations the minimal L1-norm solution is also the sparsest solution. Communications on Pure and Applied Mathematics, 59(6):797–829, 2006. ISSN 00103640. doi: 10.1002/cpa.20132.

Weinan E. A proposal on machine learning via dynamical systems. Communications in Mathematics and Statistics, 5(1):1–11, Mar 2017. ISSN 2194-671X. doi: 10.1007/s40304-017-0103-z. URL https://doi.org/10.1007/s40304-017-0103-z.

Weinan E, Chao Ma, and Lei Wu. Machine learning from a continuous viewpoint, I. Science China Mathematics, 63(11):2233–2266, nov 2020. ISSN 1674-7283. doi: 10.1007/s11425-020-1773-8.

Mehdi Fallahnezhad, Mohammad Hassan Moradi, and Salman Zaferanlouei. A Hybrid Higher Order Neural Classifier for handling classification problems. Expert Systems with Applications, 38(1):386–393, 2011. ISSN 09574174. doi: 10.1016/j.eswa.2010.06.077. URL http://dx.doi.org/10.1016/j.eswa.2010.06.077.

C. F. Fung, S.A. Billings, and H. Zhang. Generalised Transfer Functions of Neural Networks. May 1996.

C. Lee Giles and Tom Maxwell. Learning, invariance, and generalization in high-order neural networks. Applied Optics, 26(23):4972, 1987. ISSN 0003-6935. doi: 10.1364/ao.26.004972.

Rafael C Gonzalez and Richard E Woods. Digital Image Processing, Global Edition. Pearson Education, London, England, 4 edition, 2017.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, International Conference on Learning Representations, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Conference on Computer Vision and Pattern Recognition, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Young William Henry. On the multiplication of successions of Fourier constants. Proceedings of the Royal Society of London., 87(596):331–339, Oct 1912. ISSN 0950-1207. doi: 10.1098/rspa.1912.0086. URL https://royalsocietypublishing.org/doi/10.1098/rspa.1912.0086.

Roger A. Horn and Charles R. Johnson. Matrix Analysis. Cambridge University Press, dec 1985. ISBN 9780521386326. doi: 10.1017/CBO9780511810817.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359–366, 1989. ISSN 08936080. doi: 10.1016/0893-6080(89)90020-8.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. International Conference on Machine Learning, 1: 448–456, 2015.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. SIAM Review, 51(3):455–500, 2009. doi: 10.1137/07070111X.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, Nov 1998. ISSN 1558-2256. doi: 10.1109/5.726791.

Tao Luo, Zhi Qin John Xu, Zheng Ma, and Yaoyu Zhang. Phase diagram for two-layer ReLU neural networks at infinite-width limit. Journal of Machine Learning Research, 22: 1–47, 2021. ISSN 15337928.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In International Conference on Machine Learning, ICML'10, pages 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Gregory Naitzat, Andrey Zhitnikov, and Lek Heng Lim. Topology of deep neural networks. Journal of Machine Learning Research, 21:1–40, 2020. ISSN 15337928.

Wilson John Rugh. Nonlinear system theory. Johns Hopkins University Press Baltimore, 1981.

Y. Shin and Joydeep Ghosh. Ridge polynomial networks. IEEE Transactions on Neural Networks, 6(3):610–622, may 1995. ISSN 10459227. doi: 10.1109/72.377967. URL http://ieeexplore.ieee.org/document/377967/.

Yoan Shin and Joydeep Ghosh. Approximation of multivariate functions using ridge polynomial networks. In Proceedings of international joint conference on neural networks, volume 2, pages 380–385, 1992.

Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor Decomposition for Signal Processing and Machine Learning. IEEE Transactions on Signal Processing, 65(13):3551–3582, jul 2017. ISSN 1053-587X. doi: 10.1109/TSP.2017.2690524. URL http://ieeexplore.ieee.org/document/7891546/.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, International Conference on Learning Representations, 2014. URL http://arxiv.org/abs/1312.6199.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Conference on Computer Vision and Pattern Recognition, pages 1–9. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298594. URL https://doi.org/10.1109/CVPR.2015.7298594.

Roman Vershynin. High-dimensional probability: an introduction with applications in data science. Cambridge University Press., 2018. URL https://www.math.uci.edu/~rvershyn/papers/HDP-book/HDP-book.pdf.

Vito Volterra. Theory of functionals and of integral and integro-differential equations. Bull. Amer. Math. Soc, 38(1):623, 1932. doi: 10.1090/S0002-9904-1932-05479-9.

Jonathan Wray and Gary G. R. Green. Calculation of the volterra kernels of non-linear dynamic systems using an artificial neural network. Biological Cybernetics, 71(3):187–195, Jul 1994. ISSN 1432-0770. doi: 10.1007/BF00202758. URL https://doi.org/10.1007/BF00202758.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, International Conference on Learning Representations, 2016. URL http://arxiv.org/abs/1511.07122.

Jinshan Zeng, Shao-Bo Lin, Yuan Yao, and Ding-Xuan Zhou. On admm in deep learning: Convergence and saturation-avoidance. Journal of Machine Learning Research, 22(199): 1–67, 2021. URL http://jmlr.org/papers/v22/20-1006.html.