Designing Universally-Approximating Deep Neural Networks: A First-Order Optimization Approach

Zhoutong Wu, Mingqing Xiao, Cong Fang, and Zhouchen Lin, *Fellow, IEEE*

Abstract—Universal approximation capability, also referred to as universality, is an important property of deep neural networks, endowing them with the potency to accurately represent the underlying target function in learning tasks. In practice, the architecture of deep neural networks largely influences the performance of the models. However, most existing methodologies for designing neural architectures, such as the heuristic manual design or neural architecture search, ignore the universal approximation property, thus losing a potential safeguard about the performance. In this paper, we propose a unified framework to design the architectures of deep neural networks with a universality guarantee based on first-order optimization algorithms, where the forward pass is interpreted as the updates of an optimization algorithm. The (explicit or implicit) network is designed by replacing each gradient term in the algorithm with a learnable module similar to a two-layer network or its derivatives Specifically, we explore the realm of width-bounded neural networks, a common practical scenario, showcasing their universality. Moreover, adding operations of normalization, downsampling, and upsampling does not hurt the universality. To the best of our knowledge, this is the first work that *width-bounded* networks with universal approximation guarantee can be designed in a principled way. Our framework can inspire a variety of neural architectures including some renowned structures such as ResNet and DenseNet, as well as novel innovations. The experimental results on image classification problems demonstrate that the newly inspired networks are competitive and surpass the baselines of ResNet, DenseNet, as well as the advanced ConvNeXt and ViT, testifying to the effectiveness of our framework.

Index Terms—Neural network design, universal approximation, first-order optimization method, skip connection, width-bounded.

1 Introduction

Deep neural networks (DNNs) have demonstrated good performance in a wide range of applications such as image classification [1], natural language processing [2], and control [3]. From a theoretical point of view, the overwhelming success of DNNs can be partly attributed to its universal approximation capability, or universality. It suggests that neural networks often have sufficient power to represent the target function of the learning problem. The Universal Approximation Theorem (see e.g. [4], [5]) can be acknowledged as one of the foundations of deep learning, which offers a *priori* performance guarantee for DNNs.

In practice, network structures can significantly influence learning efficiency. An adept architecture not only encapsulates pertinent prior knowledge but also stabilizes training processes. Consequently, developing an effective architecture of neural networks has become one of the core research topics in deep learning. However, most existing methodologies for designing the architecture simply ignore the universal approximation property, thus discarding the apriori performance safeguard. While it is known that a two-layer fully-connected neural network (FCNN) achieves universality as its width goes to infinity [4], [5], practical considerations entail finite-width networks with moderate depths. Unfortunately, limited results about universality are established for such networks except for FCNN [6] and ResNet [7].

Concerning both the importance of neural network design and universal approximation, we naturally raise the following question:

Is there a principled way to design universally-approximating deep neural networks?

Answering the above question is the central interest of this paper and we strengthen the significance in two-folds: 1) It can deepen our understanding of the relation between network structure and the corresponding expressive power. 2) Such a methodology, if exists, can help design neural networks that work for very generic and complex problems for which commonly used neural networks may be inadequate.

The question we raise above has not been systematically studied before. We first review some existing network design methods including manual design [8], [9], [10], [11], [12], neural architecture search (NAS) [13], [14], and

- *Z. Wu is with Center for Data Science, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100871, P.R. China. E-mail: 2201213259@stu.pku.edu.cn*
- *M. Xiao is with National Key Lab of General AI, School of Intelligence Science and Technology, Peking University, Beijing 100871, P.R. China. E-mail: mingqing_xiao@pku.edu.cn*
- *C. Fang is with National Key Lab of General AI, School of Intelligence Science and Technology, Peking University, Beijing 100871, P.R. China, and also with the Institute for Artificial Intelligence, Peking University, Beijing 100871, P.R.China. E-mail: fangcong@pku.edu.cn (co-corresponding author)*
- *Z. Lin is with National Key Lab of General AI, School of Intelligence Science and Technology, Peking University, Beijing 100871, P.R. China, also with the Institute for Artificial Intelligence, Peking University, Beijing 100871, P.R. China, and also with Peng Cheng Laboratory, Guangzhou, P.R. China. E-mail: zlin@pku.edu.cn (co-corresponding author)*

optimization-based design [15], [16], [17], and show some inspiration. Manual design has brought about various graceful architectures such as ResNet [10] and DenseNet [11]. Technically, manual design requires the designer to have a profound understanding of network and learning tasks. The network architecture is often invented in a heuristic manner so it would be difficult to incorporate the universal approximation property. NAS searches the neural network architecture automatically, which reduces the labor costs of the designer. However, because NAS is a black-box procedure, it is not easy to pre-control the desired properties, including the universal approximation, in the current paradigm.

Recently, designing neural networks using optimization algorithms has been extensively studied. One major technique of the optimization-based design is unrolling [17]. Given an optimization problem, the essence of unrolling is to optimize the problem with an iterative algorithm and map each update to a layer of the network. The optimization problem can be accessible to the designer such as the sparse coding problem [15]. It can also be hypothetically established by the designer as an intermedium to be unrolled from [18], [19]. The optimization-based design has better interpretability. It is easier for designers to incorporate prior knowledge into the network structure. In some applications, the networks derived from optimization-based methods have achieved outstanding performance [17].

In this paper, building upon optimization-based design, we propose a unified framework to design DNNs with universal approximation. This affirmatively answers the aforementioned question. In the framework, we design from a first-order optimization algorithm and derive networks by replacing each gradient term in the algorithm with a learnable network module. Note here the explicit form of the optimization problem is *not* required. Instead, we focus on generic first-order algorithms, demonstrating that a large variety of first-order algorithms for convex optimization can inspire width-bounded neural networks with universality, i,e., the network is universal and *the width of the network is bounded by a constant independent of the precision of approximation*. Such a manner enriches the class of networks under optimization-based design. We call the rich class of the inspired network OptDNN. With our framework, one can easily obtain new architectures by applying new first-order optimization algorithms, which can have different yet equivalent reformulations.

In our framework, we do not necessarily need specific optimization *problems* to solve in order to unroll the iteration process. This addresses major issues in previous unrolling methods including 1) The optimization problem usually restricts the structure of the derived networks, since unrolling is limited to specific problems and the algorithms that solve them. 2) When the optimization problem is not explicitly given, a hypothetical problem has to be established, which usually requires strong assumptions. Our *algorithm*-inspired method is more flexible and our design rule is straightaway.

We further prove the universality of the OptDNN when it is generated from a first-order algorithm under mild conditions. Specifically, we show that with commonly-used activation functions, L^p-universality[1] is achievable for functions in $C([0,1]^d, \mathbb{R}^m)$ when $d \geq \min\{m, 2\}$ and $1 \leq p < \infty$. Our main idea is to show that OptDNNs are (more or) as expressive as neural ordinary differential equations (NODEs), which have been proven to be L^p-universal [20] To realize the idea, the new technique is to introduce a variant of linear multi-step methods (LMMs) [21], which is a discretization method for ordinary differential equations. We show that OptDNNs can be viewed as discretizations of NODEs using this method under mild conditions. By showing the convergence and feasibility of such discretizations, we assert that OptDNNs maintain enough expressive power as NODEs. We also give quantitative results of the approximation rate of NODEs or OptDNNs. It is worth noting that the universality of OptDNN holds in a *width-bounded* setting. To the best of our knowledge, this is the first work that width-bounded neural networks with universality can be designed in a systematic way.

Our framework can inspire some renowned structures such as ResNet [10] and DenseNet [11]. Moreover, we design several new networks of OptDNN using optimization algorithms such as the accelerated gradient descent [22] and the proximal gradient descent [23]. We first experimentally validate the universality of OptDNNs by nested ring separation and function approximation problems. Then we evaluate the effectiveness of our proposed neural networks by image classification problems. The results demonstrate that the newly inspired networks are very competitive with the baselines of ResNet and DenseNet. For example, on CIFAR-100, nearly all inspired networks achieve lower classification error rates compared with the counterpart Pre-activation ResNet [24]. Our framework can also inspire implicit networks [25], [26] using the ideas of proximal algorithms [23]. We also design several implicit networks and show that they also achieve competitive performance compared with the existing ones. Moreover, our framework is orthogonal to the module design prevalent in previous works and can seamlessly integrate advanced modules such as those of ConvNeXt [27] and ViT [28]. Experiments show that our new model can surpass them on ImageNet under the same setting, especially under the isotropic structure, demonstrating the potential of our method.

In summary, our contributions are as follows:

(1) We propose a unified framework to design universally-approximating width-bounded DNNs using first-order optimization algorithms. Here, we do not need to specify the optimization problem as previous optimization-based methods did.
(2) We prove that the OptDNNs inspired by convergent first-order algorithms can be universal approximators under mild conditions, which is the first universal approximation result for width-bounded networks with general skip connections. Our technique bridges the gap between the expressive power of NODEs and OptDNNs.
(3) Using our framework, we design several new architectures and showcase their universality experimentally. We further show by experiments that the proposed

1. The distance of two functions (or vector-valued) functions is measured in the L^p-norm.

architectures are competitive and surpass the baselines of ResNet, DenseNet, as well as the advanced ConvNeXt and ViT.

The rest of the paper is organized as follows. Section 2 reviews the studies related to unrolling and the universality of neural networks. Section 3 and 4 answer the main question, where in Section 3 we present our design rules as well as some examples of designing networks, and in Section 4 we prove the universality of our OptDNNs under mild conditions. Section 5 conducts experiments on OptDNNs and evaluates their performances on different problems. Section 6 concludes the paper with future directions.

2 Related Works

In this section, we review the literature of unrolling and the universality of neural networks, which we believe is most related to our work. We will compare our work with the highly related ones and show the difference and advantages.

2.1 Unrolling

Unrolling is one of the major techniques of the optimization-based design methods. To design networks by unrolling, the designer usually starts from an optimization problem with an explicit objective function. The DNNs can be obtained by unrolling the iterative process that solves the problem [15], [16], [33], [34], [35].

Usually, the explicit optimization problem is accessible to the designer, such as the sparse coding or compress sensing problem. The earliest network may be the LISTA network from [15] which solves the LASSO problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{y}-\mathbf{W}\mathbf{x}\|_2^2+\lambda\|\mathbf{x}\|_1$$

with updates

$$\mathbf{x}^{k+1}=S_\lambda\left(\left(\mathbf{I}-\frac{1}{\mu}\mathbf{W}^T\mathbf{W}\right)\mathbf{x}^k+\frac{1}{\mu}\mathbf{W}^T\mathbf{y}\right), \qquad (1)$$

where S_λ is the soft-thresholding operator. The iteration in Eq. (1) resembles a single layer of a feedforward network when S_λ is viewed as the activation function and $\mathbf{W}$ as the weight. Then the LISTA network is derived by mapping Eq. (1) to a network layer and stacking a fixed number of layers together. Moreover, the parameters in the LISTA network are learned from training samples, and the weight can vary in different layers. Because unrolling requires that the optimization updates can be mapped into a network layer (like Eq. (1)), it is restricted to specific problems and the corresponding optimization algorithms, which limits the capacity of inspired network architectures.

When the explicit optimization problem is not given, some works attempted to first establish a hypothetical optimization problem and then design networks from it. For example, Li et al. [18], Bibi et al. [19] and Shen et al. [36] showed that, under some conditions, the forward pass in FCNN is equivalent to the iteration of optimizing a problem using specific algorithms. They then designed new architectures by solving the hypothetical problem using different algorithms and unrolling the iterative processes. Yet in the modeling process, the authors assumed that the weight matrix is symmetric and positive semi-definite to make the iteration a gradient descent of an objective function, which is inconsistent with real situations. There is also a lack of guidance to construct the hypothetical problem if without enough assumptions.

Our design method generalizes these design methods since ours does not stick to specific explicit problems. Moreover, our method remains applicable when we have an explicit optimization problem. Therefore, our method is more flexible and can be used in broad settings.

2.2 The Universality of Neural Networks

The universality of neural networks was first studied in [4] and [5], where the authors proved that two-layer[2] FCNN with a discriminatory activation function is universal in the continuous function space. Extensions to non-polynomial activation functions followed [37]. These classical works focus on neural networks of bounded depth and unbounded width. That means the width would grow to infinity when the approximation error goes to 0. Recent research delves into networks with bounded width and unbounded depth, aligning more closely with real-world scenarios. Table 1 summarizes existing results for width-bounded networks alongside our OptDNNs. We can see that most of the existing analysis only focuses on the standard FCNN, whereas, the OptDNNs are rich and include many kinds of DNNs.

Another research line is to study the universal approximation of networks with skip connections. Although skip connections are widely used in modern neural networks, there are limited works on the expressive power of networks with skip connections. Lin et al. [7] proved the L^1-universality of width-bounded ReLU ResNet. Sanders et al. [38] showed that the momentum residual network (MResNet) can learn any linear mappings up to a multiplicative factor. The most related work to ours is [20], where the authors studied the approximation properties of ResNet from the perspective of dynamic system. They proved the $L^p(K,\mathbb{R}^m)$-universality of the continuous ResNet, i.e., the NODE for $1\leq p<\infty$ when $d\geq 2$, where $K\subset\mathbb{R}^d$ is any compact set. The major differences between the present paper and [20] are summarized as follows. 1) The starting point is different. In [20], the author focused on the approximation ability of NODEs. In contrast, we aim at proposing a unified framework to design networks with universality guarantee. 2) In [20], the authors only focused on continuous-time networks, i.e., NODEs. In contrast, we show the expressive power of the layer-based OptDNNs. To achieve our results, we propose to use a variant of the LMM to discretize NODEs. So our work further bridges the gap between the expressive power of the NODEs and layer-based networks with skip connections. 3) In [20], the authors only give the approximation rate of NODEs when the input dimension $d=1$ for a small class of functions, whereas we use new techniques to give the approximation rate of NODEs and also OptDNNs in a much more general setting that $d>\max\{1,m\}$ for all continuous functions.

3 Unified Framework of Network Design

In this section, we introduce our unified framework and provide some examples to elucidate how to concretely de-

2. The input layer is not counted in.

TABLE 1
Summary of the known results of the universality of width-bounded networks. In the table, we denote by d and m the input and output dimensions of the network, respectively. $K \subset \mathbb{R}^d$ denotes a compact set and $1 \leq p < \infty$. w denotes the minimum width of the network for universal approximation. Note that the OptDNNs have rich structures and include ResNet as a special case.

Reference	Function Class	Structure	Activation Function	Conditions on Minimal Width
Lu et al. [6]	$L^1(\mathbb{R}^d, \mathbb{R})$	FCNN	ReLU	$d+1 \leq w \leq d+4$
	$L^1(K, \mathbb{R})$	FCNN	ReLU	$w \geq d$
Johnson [29]	$C(K, \mathbb{R})$	FCNN	Uniformly Continuous	$w \geq d+1$
Hanin et al. [30]	$C(K, \mathbb{R}^m)$	FCNN	ReLU	$d+1 \leq w \leq d+m$
Park et al. [31]	$L^p(\mathbb{R}^d, \mathbb{R}^m)$	FCNN	ReLU	$w = \max\{d+1, m\}$
	$C(K, \mathbb{R}^m)$	FCNN	ReLU & Step	$w = \max\{d+1, m\}$
Kidger et al. [32]	$C(K, \mathbb{R}^m)$	FCNN	Nonaffine Polynomial	$w \leq d+m+2$
Lin et al. [7]	$L^1(\mathbb{R}^d, \mathbb{R})$	ResNet	ReLU	$w \leq d$
Li et al. [20]	$L^p(K, \mathbb{R}^m)$	ResNet[1]	ReLU & Sigmoid	$w \leq d$
Ours	$L^p(K, \mathbb{R}^m)$	OptDNNs	Commonly Used[2]	$w \leq d$

[1] It is a continuous-time ResNet, to be precise.
[2] Examples of the activation functions for OptDNNs can be found in Table 3.

TABLE 2
Summary of the notations in this paper. The learnable module T can be a convolution operator or a residual block in the network.

Notation	Definition
$\mathbf{z}_0$	The input of a neural network
$\mathbf{z}_k$	The output of the k-th layer of a neural network
σ	A general activation function
σ_R	The ReLU activation function
T	A general learnable module in a neural network
$\lceil x \rceil$	The smallest integer no less than the scalar x
δ	The Kronecker delta
$\mathbf{1}$	The indicator function
$\mathrm{Prox}_{\lambda f}$	The (norm-2) proximal operator of function f
$\mathscr{A}$	An iterative optimization algorithm
$\mathbf{I}_k$	The $k \times k$ identity matrix
$\mathbf{P}(i,j)$	The entry of matrix $\mathbf{P}$ at row i and column j
$\otimes$	The Kronecker product of matrices
$\|\cdot\|$	A general norm of a vector or a matrix
$\|\cdot\|_p$	The ℓ^p vector norm or the subordinate matrix norm
$\overline{S}$	The closure of a set S under usual topology
$\mathrm{CH}(S)$	The convex hull of a set S
K	A compact set in $\mathbb{R}^d$
$C(K)$	The set of continuous functions on K
$C(K, \mathbb{R}^m)$	The set of continuous functions from K to $\mathbb{R}^m$
$\|h\|_{L^p(K)}$	The L^p-norm of $h \in L^p(K)$, i.e., $\left(\int_K |h|^p\right)^{\frac{1}{p}}$
$\|g\|_{L^p(K,\mathbb{R}^m)}$	The L^p-norm of $g \in L^p(K, \mathbb{R}^m)$
g_i	The i-th entry of a vector-valued function g

sign neural networks from different algorithms.

3.1 Preliminaries

We summarize the main notations used in this paper in Table 2. We give some useful definitions and assumptions as follows.

Definition 1 (Lipschitz continuous function)**.** *A function $g \in C(K)$ is Lipschitz continuous if there exists a constant $L_g > 0$ such that*

$$|g(\mathbf{x}) - g(\mathbf{y})| \leq L_g \|\mathbf{x} - \mathbf{y}\|_\infty$$

for all $\mathbf{x}, \mathbf{y} \in K$. In this case, we also say that g is L_g-Lipschitz. Similarly, a vector-valued function $g \in C(K, \mathbb{R}^m)$ is Lipschitz continuous if there exists a constant L_g (L_g-Lipschitz) such that

$$\|g(\mathbf{x}) - g(\mathbf{y})\|_\infty \leq L_g \|\mathbf{x} - \mathbf{y}\|_\infty$$

for all $\mathbf{x}, \mathbf{y} \in K$.

TABLE 3
Some activation functions that Assumption 1 holds for. We denote the cumulative distribution function of the Gaussian distribution $\mathcal{N}(0,1)$ by $\Phi(\cdot)$. It can be seen that many commonly used activation functions, such as ReLU, Sigmoid, and Tanh [39] satisfy Assumption 1.

	Activation function		Activation function
Sigmoid	$\frac{1}{1+e^{-x}}$	Softsign	$\frac{x}{1+\lvert x\rvert}$
Tanh	$\frac{1-e^{-2x}}{1+e^{-2x}}$	Swish	$\frac{x}{1+e^{-x}}$
Softplus	$\log(e^x+1)$	GELU	$x\Phi(x)$
ReLU	$\begin{cases} x, & x \geq 0, \\ 0, & x < 0. \end{cases}$	ELU	$\begin{cases} x, & x \geq 0, \\ \alpha(e^x - 1), & x < 0. \end{cases}$
Leaky ReLU	$\begin{cases} x, & x \geq 0, \\ \alpha x, & x < 0. \end{cases}$	SELU	$\begin{cases} \lambda x, & x \geq 0, \\ \lambda\alpha(e^x - 1), & x < 0. \end{cases}$

Definition 2 (Lipschitz family)**.** *A set of functions $\mathcal{F}$ is called a Lipschitz family if for any $\mathbf{u} \in \mathcal{F}$, $\mathbf{u}$ is Lipschitz continuous.*

Definition 3 (Control family)**.** *For a family of ODEs*

$$\{\dot{\mathbf{x}}(t) = \mathbf{u}(\mathbf{x}(t)) : \mathbf{u} \in \mathcal{F}\},$$

we call $\mathcal{F}$ a control family since it controls the flow map induced by the above ODEs.

Assumption 1. *The activation function $\sigma : \mathbb{R} \to \mathbb{R}$ satisfies the following conditions:*

1) *σ is L_σ-Lipschitz on $\mathbb{R}$.*
2) *σ is infinitely differentiable on $\mathbb{R}$ except for at most finitely many points.*
3) *$\forall \varepsilon > 0$, there exist $a, b \in \mathbb{R}$ with $a < b$ and affine functions $v_1(x)$ and $v_2(x)$ such that*

$$\|\sigma - v_1\|_{C((-\infty,a])} \leq \varepsilon, \quad \|\sigma - v_2\|_{C([b,\infty))} \leq \varepsilon.$$

Assumption 1 for activation function is very weak which can be satisfied by most commonly used activation functions. Here, some examples are provided in Table 3.

3.2 Design Neural Networks with First-Order Optimization Algorithms

We now show how to design neural architectures using first-order optimization algorithms. Consider a generic smooth convex optimization problem

$$\min_{\mathbf{x} \in \mathbb{R}^d} f(\mathbf{x}).$$

We consider a first-order algorithm solving the above problem whose updates can be formulated as

$$\mathbf{x}_{k+1} = \sum_{i=0}^{k} \alpha_i^{k+1} \mathbf{x}_i + \sum_{i=0}^{k+1} \beta_i^{k+1} \nabla f(\mathbf{x}_i), \tag{2}$$

where $\{\alpha_i^k\}_{k,i}$ and $\{\beta_i^k\}_{k,i}$ are some pre-defined hyper-parameters. Note that update (2) includes a variety of first-order algorithms. For example, it includes the proximal algorithm since we have

$$\mathbf{x}_{k+1} = \mathrm{Prox}_{\lambda f}(\mathbf{x}_k) \iff \mathbf{x}_{k+1} = \mathbf{x}_k - \lambda \nabla f(\mathbf{x}_{k+1}), \tag{3}$$

for $\lambda > 0$, where

$$\mathrm{Prox}_{\lambda f}(\mathbf{x}) = \underset{\mathbf{y}}{\mathrm{argmin}} \left(f(\mathbf{y}) + \frac{1}{2\lambda} \|\mathbf{y} - \mathbf{x}\|_2^2 \right).$$

It also includes some algorithms that use auxiliary variables. For example, for a gradient-based algorithm with two inputs $\mathbf{x}^0$ and $\mathbf{y}^0$ that updates $\mathbf{x}$ and $\mathbf{y}$ simultaneously, one can treat the updates in a serial manner as $\mathbf{x}^0, \mathbf{y}^0, \mathbf{x}^1, \mathbf{y}^1 \cdots$. Since we do not require α_i^k and β_i^k to be nonzero, this algorithm can be included in Eq. (2).

In our framework, we design OptDNN from Eq.(2) by replacing the gradient terms with learnable modules T in the network

$$\mathbf{z}_{k+1} = \sum_{i=0}^{k} \alpha_i^{k+1} \mathbf{z}_i + \sum_{i=0}^{k+1} \beta_i^{k+1} T_i^{k+1}(\mathbf{z}_i), \quad k = 0, \cdots, L-1. \tag{4}$$

We first set

$$T_i^k(\mathbf{z}) = \mathbf{V}_i^k \sigma(\mathbf{W}_i^k \mathbf{z} + \mathbf{b}_i^k), \tag{5}$$

where $\mathbf{W}^k \in \mathbb{R}^{q \times d}$, $\mathbf{V}^k \in \mathbb{R}^{d \times q}$, and $\mathbf{b}^k \in \mathbb{R}^q$ are the weight matrix and bias, respectively, and the activation function σ satisfies Assumption 1. We will also discuss other forms of T (See Section 5). Note here we do not require that two equivalent algorithms inspire the same network. We call the coefficients α_i^k and β_i^k the modulating parameters in this paper. For the update of different layers, the modulating parameters are untied as k is used both as the index of layers and the number of steps, and we allow $\mathbf{z}_{k+1}$ to have skip connections with all previous layers (in practice, the number of skip connections are usually bounded by a constant). In the last layer of the network, an affine transformation is applied

$$\mathbf{z}_{L+1} = \mathbf{W}^{L+1} \mathbf{z}_L + \mathbf{b}^{L+1}. \tag{6}$$

An illustration of our design method is provided in Figure 1.

In Eq. (4), when $\beta_i^{k+1} = 0$, the inspired network is an explicit feedforward network and the network parameters can be learned using the standard back-propagation algorithm. When $\beta_i^{k+1} \neq 0$, we obtain an implicit network, where the output values of some individual layers are solutions to some fixed-point equations. The implicit network [25], [26] generalizes the recursive update rule in FCNNs and owns some special properties and advantages. For instance, some implicit networks can reduce memory consumption [25], enhance robustness [40], and introduce prior knowledge or properties into the model [41]. For the implicit network designed in our framework, we can use a fixed-point method [40] to approximate the implicit scheme or use an implicit differentiation method [25], [26] to update the weights.

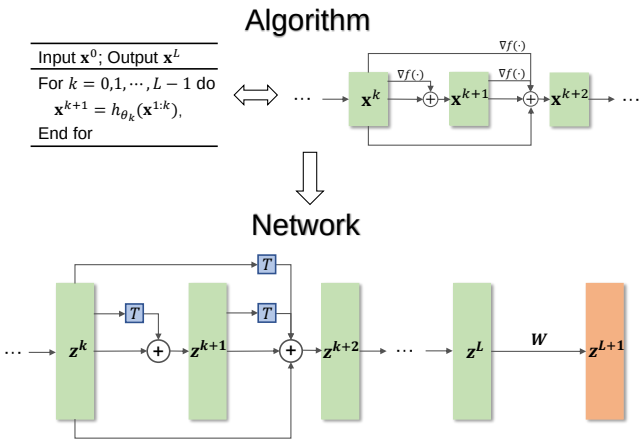


Fig. 1. Our framework for designing neural architectures. We can design from serial-update gradient-based optimization algorithms, including the algorithms that can be transformed in a serial-update manner. The gradient terms in each update of the optimization algorithm are replaced with T in the inspired network. And an affine transformation is applied in the last layer of the network.

It is known that a large number of optimization algorithms can be viewed as discretizations of gradient-ODEs. Consequently, some networks designed within our framework can also be derived from specific ODE discretization schemes. This connection can be helpful as demonstrated in our proof of the universality of OptDNNs in Section 4 . As an alternative, one can also understand our framework from an ODE-based perspective. In this paper, we choose to introduce our framework from an optimization standpoint because the rich class of optimization algorithms can inspire more network architectures, in our view. Additionally, this perspective renders our framework applicable when there is a given explicit objective function for the learning problem. To provide a comprehensive understanding, we reference several works related to ODE-based design [40], [42], [43].

3.3 Examples of OptDNN

In this subsection, we will give some examples of OptDNN to concretely show how to design neural networks using modern first-order optimization algorithms.

Gradient Descent. Consider minimizing $f(\mathbf{x})$. In our framework, gradient descent

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \beta \nabla f(\mathbf{x}_k)$$

directly inspires ResNet[3]:

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \beta T^k(\mathbf{z}_k).$$

3. It is a bit different from the original ResNet in [10]. However, we can derive the commonly used ResNet by replacing the module T in Eq. (5) with the residual units in [24]. In fact, we can consider many different forms of module T in our framework and we will discuss it in Section 5.

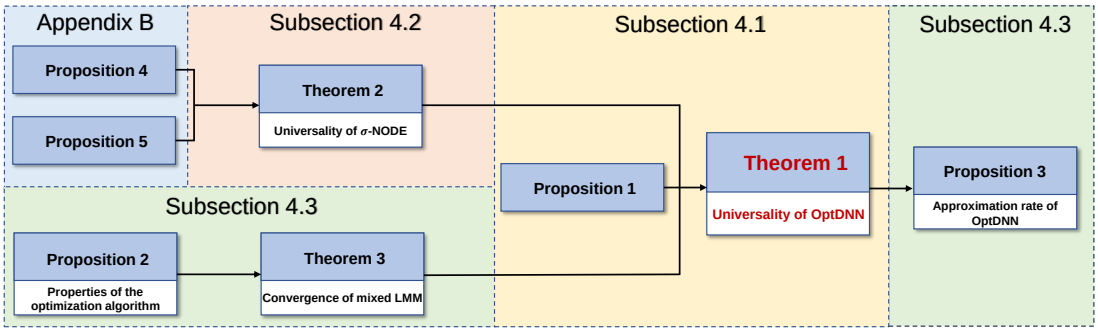


Fig. 2. An illustration of the line of the theoretical results. We use $A \to B$ to indicate that B is proved based on the result or proof technique of A. The background knowledge of Subsection 4.3 is given in Appendix A.

Accelerated Gradient Descent [22]. Consider minimizing $f(\mathbf{x})$. One accelerated gradient descent method consists of iterations

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \sum_{j=0}^{k} h_{k+1,j} \nabla f(\mathbf{x}_j),$$

where $\{h_{k,j}\}$ are some fixed parameters (see Appendix C.1). The inspired network can be represented as:

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \sum_{j=0}^{k} h_{k+1,j} T_j^k(\mathbf{z}_j). \tag{7}$$

This network can be regarded as a version of DenseNet

$$\mathbf{z}_{k+1} = T^k([\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_k]),$$

where $[\cdot]$ refers to the concatenation operation.

The Heavy Ball Algorithm [44]. Still consider minimizing $f(\mathbf{x})$. The heavy ball algorithm consists of iterations

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \beta \nabla f(\mathbf{x}_k) + \alpha(\mathbf{x}_k - \mathbf{x}_{k-1}). \tag{8}$$

Using our method, the inspired network can be represented as

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \beta T^k(\mathbf{z}_k) + \alpha(\mathbf{z}_k - \mathbf{z}_{k-1}), \tag{9}$$

which gives the HB-NN in [18]. Besides, by introducing auxiliary variables $\mathbf{y}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$ in Eq. (8), an equivalent form of the heavy ball algorithm is

$$\begin{aligned} \mathbf{y}_{k+1} &= \alpha \mathbf{y}_k - \beta \nabla f(\mathbf{x}_k), \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{y}_{k+1}. \end{aligned} \tag{10}$$

Then the network inspired by Eq. (10) can be represented as

$$\begin{aligned} \mathbf{y}_{k+1} &= \alpha \mathbf{y}_k + \beta T^k(\mathbf{z}_k), \\ \mathbf{z}_{k+1} &= \mathbf{z}_k + \mathbf{y}_{k+1}. \end{aligned}$$

This corresponds to the MResNet in [38]. Although the HB-NN and the MResNet are inspired by the equivalent forms of the same algorithm, they perform differently in experiments. This also implies the flexibility of our framework.

Linearized ADMM [45]. The linearized ADMM is used to solve constrained optimization problems. Similarly, we consider minimizing $f(\mathbf{x})$ and write it equivalently as the following problem:

$$\min_{\mathbf{x},\mathbf{y}} f_1(\mathbf{x}) + f_2(\mathbf{y}), \quad s.t. \quad \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{b}, \tag{11}$$

where we set $f_1 = f_2 = f$ and $\mathbf{A} = \mathbf{I}, \mathbf{B} = -\mathbf{I}, \mathbf{b} = \mathbf{0}$. Then the iterations of one sort of Linearized ADMM can be expressed as

$$\begin{aligned} \mathbf{x}_{k+1} &= \frac{1}{2}\left(\nabla f(\mathbf{x}_k) + \mathbf{y}_k - \sum_{t=1}^{k}(\mathbf{x}_t - \mathbf{y}_t)\right), \\ \mathbf{y}_{k+1} &= \frac{1}{2}\left(\nabla f(\mathbf{y}_k) + \mathbf{x}_{k+1} + \sum_{t=1}^{k}(\mathbf{x}_t - \mathbf{y}_t)\right). \end{aligned} \tag{12}$$

Thus the inspired network can be expressed as

$$\begin{aligned} \mathbf{x}_{k+1} &= \frac{1}{2}\left(T^k(\mathbf{x}_k) + \mathbf{y}_k - \sum_{t=1}^{k}(\mathbf{x}_t - \mathbf{y}_t)\right), \\ \mathbf{y}_{k+1} &= \frac{1}{2}\left(T^k(\mathbf{y}_k) + \mathbf{x}_{k+1} + \sum_{t=1}^{k}(\mathbf{x}_t - \mathbf{y}_t)\right). \end{aligned}$$

Proximal Gradient Descent [23]. The proximal gradient descent (PGD) is used to solve the composite optimization problem, namely, $\min_{\mathbf{x}} f_1(\mathbf{x}) + f_2(\mathbf{x})$. In order to use our method to derive a network from PGD, we still consider minimizing $f(\mathbf{x})$ and equivalently write it as a composite optimization problem

$$\min_{\mathbf{x}} f(\mathbf{x}) = \underbrace{f(\mathbf{x})}_{f_1(\mathbf{x})} + \underbrace{f(\mathbf{x})}_{f_2(\mathbf{x})}. \tag{13}$$

PGD consists of iterations

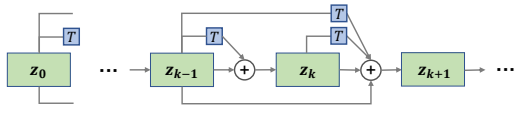
$$\begin{aligned} \mathbf{y}_k &= \mathbf{x}_k - t_k \nabla f_2(\mathbf{x}_k), \\ \mathbf{x}_{k+1} &= \mathrm{Prox}_{t_k f_1}(\mathbf{y}_k), \end{aligned}$$

where t_k is the step size. For simplicity, we take $t_k = 1$ and the inspired network can be represented as

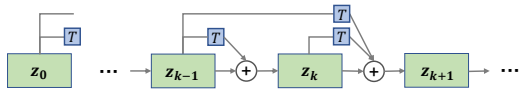
$$\begin{aligned} \mathbf{y}_k &= \mathbf{x}_k + T_1^k(\mathbf{x}_k), \\ \mathbf{z}_{k+1} &= \mathbf{y}_k + T_2^k(\mathbf{z}_{k+1}). \end{aligned}$$

This network is an implicit network.

From the above examples, we see that our methods can consider some composite optimization algorithms, as well as some constrained optimization algorithms. In addition to the above examples, more details and examples of OptDNN are presented in Appendix C.1. The structures of the inspired network above are provided in Subsection 3.4 and Appendix C.1.



(a) The architecture of the AGD-NN in [18].



(b) The architecture of our OptDNN-AGD.

Fig. 3. Demonstration of the structures of AGD-NN in [18] and the network inspired by accelerated gradient using our method (OptDNN-AGD). Note that in AGD-NN, $\mathbf{z}_k$ has residual connections to $\mathbf{z}_0, \cdots, \mathbf{z}_{k-1}$, while in our OptDNN-AGD, $\mathbf{z}_k$ only has a residual connection to $\mathbf{z}_{k-1}$.

3.4 More Differences from Previous Works

In addition to the differences in the overall methodology that we have described in Sections 1 and 2.1, in this subsection, we emphasize more differences between our design method and those of previous works, especially in [18] in details of network architectures, and show the superiority of our framework.

In fact, even inspired by the same optimization algorithm, the resulting networks are different. For example, [18] also designed networks from the heavy-ball algorithm and the accelerated proximal gradient, resulting in HB-NN

$$\mathbf{z}_{k+1} = T^k(\mathbf{z}_k) + \alpha(\mathbf{z}_k - \mathbf{z}_{k-1}),$$

and AGD-NN

$$\mathbf{z}_{k+1} = \sum_{j=0}^{k} h_{k+1,j} T^k(\mathbf{z}_j) + \mathbf{z}_k - \sum_{j=0}^{k} h_{k+1,j}\mathbf{z}_j,$$

respectively. The HB-NN is different from its counterpart Eq. (9) as one more current value term $\mathbf{z}_k$ is added in the calculation of $\mathbf{z}_{k+1}$ in Eq. (9), which is similar to the difference between FCNN and ResNet. Specifically, to construct identity mapping by an added layer, i.e., to make $\mathbf{z}_{k+1} = \mathbf{z}_k$, T in HB-NN needs to approximate the identity mapping while T in Eq. (9) needs to approximate zero if $\mathbf{z}_k$ is close to $\mathbf{z}_{k-1}$. The AGD-NN is clearly different from the counterpart Eq. (7) in skip connections. An illustration is provided in Figure 3. This is because [18] has to establish a hypothetical objective function and derive the network architecture following the computation process, after applying an optimization algorithm to the hypothesized objective function, while ours simply replaces the gradient terms in the optimization algorithm with T modules Such difference in derivation causes the difference in the resulting network architectures. In order to obtain the hypothesized objective function, [18] has to assume that the weight matrix be symmetric and positive semi-definite and be fixed across different layers, which is unrealistic. Moreover, writing down the details of applying an optimization algorithm to the hypothesized objective function requires some effort and accordingly, operations such as downsampling, upsampling and normalization cannot be included as they are not part of the optimization algorithm[4]. In contrast, the T modules in our framework can include these extra operations (See Section 5). So our framework is much more flexible and easier to use. More importantly, we can prove that under mild conditions the networks designed by our framework are universal, while those in [18] do not have such a guarantee.

4 The Universality of OptDNN

In this section, we prove that the OptDNN inspired by any convergent first-order optimization algorithm can be a universal approximator under some conditions on the parameters. The line of proving the universality results is illustrated in Figure 2. In Subsection 4.1, we present the main theorem. In Subsection 4.2, we present the results of the universality of the first-order NODE

$$\dot{\mathbf{z}}(t) = \mathbf{u}(t, \mathbf{z}(t)), \quad \mathbf{z}(0) = \mathbf{z}_0, \tag{14}$$

where $\mathbf{u}(t, \mathbf{z}) = \mathbf{V}^t\sigma(\mathbf{W}^t\mathbf{z} + \mathbf{b}^t)$. In subsection 4.3, we propose a convergent discretization method that we call 'mixed' LMM and show that OptDNNs are at least as expressive as NODEs based on this method, from which we can prove the main theorem. We also give the approximation rate of OptDNNs as well as NODEs under mild conditions.

4.1 Main Results

Recall that OptDNN can be represented as in Eq. (4) and Eq. (6). We set

$$T^k(\mathbf{z}) = \mathbf{V}^k\sigma(\mathbf{W}^k\mathbf{z} + \mathbf{b}^k),$$

where $\mathbf{W}^k \in \mathbb{R}^{q\times d}, \mathbf{V}^k \in \mathbb{R}^{d\times q}$, and $\mathbf{b}^k \in \mathbb{R}^q$ and Assumption 1 holds. We consider $q = 3$ in our analysis for simplicity. The analysis can be easily extended to $q > 3$.To show the universality on the compact set K, we set K as $[0, 1]^d$ without loss of generality.

Usually, the hyper-parameters in the optimization algorithm, such as the step length or momentum coefficients, are adaptive. To prove the universality, we need some conditions on the parameters as shown in Assumption 2.

Assumption 2. *Assume that the considered optimization algorithm $\mathscr{A}$ for solving any smooth, convex, unconstrained optimization problem $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$ can be represented as in Eq.* (2)*. The algorithm further admits the following conditions*

1) *For any k, $\sum_{i=0}^{k} \alpha_i^{k+1} = 1$.*
2) *There exists $k_0 > 0$, such that for any $k \geq k_0$, it holds that $\alpha_j^k = 0, \forall j \leq k - k_0$, and $\sum_{i=k_0}^{k} |\beta_i^k| \neq 0$.*
3) *There exists $\varepsilon > 0$, such that for any k, except for at most one root 1, all other roots of the polynomial $\rho_k(\lambda) = \sum_{i=0}^{k} \alpha_i^k \lambda^i$ lies in the open unit disk of radius $1 - \varepsilon$.*

Assumption 2 is usually satisfied for algorithms under common parameter choices. For example, the heavy ball algorithm described in Eq. (8) satisfies Assumption 2 with the common choice of $\beta \in (0, 1)$. More examples are provided in Appendix C. In fact, condition 1) is a standard condition for optimization algorithms. Condition 2) is satisfied if $\mathscr{A}$

4. These operations and some other modifications can only be added as pure engineering tricks in [18].

updates $\mathbf{x}_k$ using only the previous k_0 iterates. Condition 3) can be roughly interpreted as that the algorithm has linear convergence when the objective function is strongly convex, which is usually satisfied from the perspective of algorithm design [46] (For more details, see Appendix B.7).

We present the main result in Theorem 1.

Theorem 1. *Let $d \geq 2$, $0 < m \leq d$, $1 \leq p < \infty$, and $\mathscr{A}$ be an optimization algorithm. If $\mathscr{A}$ satisfies Assumption 2, then the algorithm-inspired network*

$$\mathbf{z}_{k+1} = \sum_{i=0}^{k} \alpha_i^{k+1} \mathbf{z}_i + \sum_{i=0}^{k+1} \beta_i^{k+1} T_i^{k+1}(\mathbf{z}_i),\ k \leq L-1, \quad (15)$$

$$\mathbf{z}_{L+1} = \mathbf{W}^{L+1}\mathbf{z}_L + \mathbf{b}^{L+1}, \quad (16)$$

is universal for $C([0,1]^d, \mathbb{R}^m)$ in norm $\|\cdot\|_{L^p([0,1]^d,\mathbb{R}^m)}$, where

$$T_i^k(\mathbf{z}) = \mathbf{V}_i^k \sigma(\mathbf{W}_i^k \mathbf{z} + \mathbf{b}_i^k).$$

Specifically, for any vector-valued function $g \in C([0,1]^d, \mathbb{R}^m)$, $\forall \varepsilon > 0$, there exists an OptDNN network in the form of Eq. (15) and Eq. (16), that the network function $h : \mathbb{R}^d \to \mathbb{R}^m$, $h(\mathbf{z}_0) = \mathbf{z}_{L+1}$ satisfies

$$\|g - h\|_{L^p([0,1]^d,\mathbb{R}^m)} \leq \varepsilon.$$

We further show that we can just prove a simplified version of Theorem 1. We can assume that $m = d$ and neglect the affine transformation of Eq. (16) at the end of the OptDNN. This is guaranteed by the following proposition.

Proposition 1. *Let $d \geq m$, $K \subset \mathbb{R}^d$ be a compact set and $l : \mathbb{R}^d \to \mathbb{R}^m$ be a linear surjection. Then for any continuous vector-valued function $\phi : K \to \mathbb{R}^m$ and $\varepsilon > 0$, there exists a continuous function $\psi : K \to \mathbb{R}^d$, such that*

$$\|\phi - l \circ \psi\|_{C(K,\mathbb{R}^m)} \leq \varepsilon.$$

Note in Proposition 1 that the surjection l exists since $d \geq m$. To approximate a continuous function $g : \mathbb{R}^d \to \mathbb{R}^m$, the affine transformation of Eq. (16) can be fixed as l in Proposition 1. If the function $\mathbf{z} \mapsto \mathbf{z}_L$ in Eq. (15) can approximate ψ, then the corresponding OptDNN can approximate g.

4.2 The Universal Approximation of First-order NODEs

In this subsection, we will first give the result of the L^p-universality of first-order NODEs based on the results in [20]. In [20], the authors prove the L^p-universality of the NODE of Eq. (14) when $\mathcal{F} := \{\mathbf{z} \mapsto \mathbf{u}(t, \mathbf{z})\}$ satisfies some topological conditions. Then they also consider $\mathbf{u}(t, \mathbf{z}) = \mathbf{V}^t \tilde{\sigma}(\mathbf{W}^t \mathbf{z} + \mathbf{b}^t)$, where $\tilde{\sigma}$ is the activation function. However, they do not give explicit conditions on $\tilde{\sigma}$ that the universality holds, so one has to examine whether $\mathcal{F}_{\tilde{\sigma}}$ satisfies their conditions for every specific $\tilde{\sigma}$, which is not easy (For instance, see Example 2.7 in [20]). As an extension, we show that universality is achievable when $\tilde{\sigma}$ satisfies Assumption 1, which includes many commonly used activation functions. The result is given in Theorem 2.

Theorem 2. *Let $d \geq 2$, $1 \leq p < \infty$, and $g : \mathbb{R}^d \to \mathbb{R}^d$ be a continuous vector-valued function. Suppose that the activation function σ satisfies the conditions in Assumption 1. Then $\forall \varepsilon > 0$, there exists a function $\mathbf{u}(t, \mathbf{z}) = \mathbf{V}^t \sigma(\mathbf{W}^t \mathbf{z} + \mathbf{b}^t)$ and $\tau > 0$, such that*

$$\|g - h\|_{L^p([0,1]^d,\mathbb{R}^d)} \leq \varepsilon,$$

where $h(\mathbf{z}_0) = \mathbf{z}(\tau; \mathbf{z}_0)$, and $\mathbf{z}$ is defined by

$$\dot{\mathbf{z}}(t) = \mathbf{u}(t, \mathbf{z}(t)), \quad \mathbf{z}(0) = 0, \quad t \in [0, \tau].$$

Remark 1. *The expressive power of the NODE has also been studied in [47], [48], [49]. In [49], the authors showed that NODEs are universal for a large class of diffeomorphisms in the sense of $\|\cdot\|_{L^\infty(K,\mathbb{R}^m)}$ when the control family is universal. In Subsection 4.3, we will show that OptDNNs are at least as expressive as NODEs. Thus OptDNNs own all the favorable approximation properties of NODEs.*

4.3 The Universal Approximation of OptDNNs

In this subsection, we will first give a brief introduction to LMM and some relevant notions. Then we will present a discretization method that we call 'mixed' LMM. OptDNNs can be derived by discretizing the NODEs with 'mixed' LMMs. We will further show that the mixed LMM is convergent, thus OptDNNs have at least the same representation capability as NODEs.

It is known that the Euler discretization of Eq. (14)

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \Delta t \mathbf{u}(t_k, \mathbf{z}_k)$$

indicates a ResNet [50]. Nevertheless, the Euler discretization cannot lead to more networks with general skip connections. We adopt the strategy to alter the discretization methods to learn the expressive power of the corresponding networks, which can have general skip connections.

4.3.1 Brief Introduction to LMM

Assume that function $\mathbf{u}(t, \mathbf{x}) : [t_0, \infty) \times \mathbb{R}^d \to \mathbb{R}^d$ is L_u-Lipschitz with respect to $\mathbf{x}$ for all t. For a well-posed ODE

$$\dot{\mathbf{x}}(t) = \mathbf{u}(t, \mathbf{x}(t)), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \quad (17)$$

the k-step LMM formula [21] is

$$\sum_{j=0}^{k} \alpha_j \mathbf{x}_{n+j} = \Delta t \sum_{j=0}^{k} \beta_j \mathbf{u}_{n+j}, \quad (18)$$

where $\mathbf{x}_j$ is the calculated value at time $t_j := t + j\Delta t$, $\mathbf{u}_j = \mathbf{u}(t + j\Delta t, \mathbf{x}_j)$, $\alpha_k \neq 0$, and $|\alpha_0| + |\beta_0| \neq 0$. Here $\mathbf{x}_0, \cdots, \mathbf{x}_{k-1}$ are some prescribed initial values. When $\beta_k = 0$, the method is explicit, otherwise it is implicit. The generating polynomials [51] of Eq. (18) are defined as

$$\rho(\lambda) = \sum_{j=0}^{k} \alpha_j \lambda^j, \quad \theta(\lambda) = \sum_{j=0}^{k} \beta_j \lambda^j. \quad (19)$$

We further introduce the concepts of consistency, stability, and convergence.

Definition 4 (Consistency [51])**.** *The linear multi-step method Eq. (18) is said to be consistent if*

$$\sum_{j=0}^{k} [\alpha_j \mathbf{x}(t_n + j\Delta t) - \Delta t \beta_j \dot{\mathbf{x}}(t_n + j\Delta t)] = \mathbf{c}\Delta t^2 + \mathcal{O}(\Delta t^3),$$

where $\mathbf{c} \in \mathbb{R}^d$ is a non-zero vector.

The consistency condition is equivalent to

$$\rho(1) = 0, \quad \dot{\rho}(1) = \theta(1). \tag{20}$$

Definition 5 (Stability [51])**.** *The linear multi-step method Eq.* (18) *is said to be stable, or more specifically, zero-stable, if the generating polynomial $\rho(\lambda)$ in Eq.(19) satisfies the root condition, i.e.,*

1) *The roots of $\rho(\lambda)$ lie on or within the unit circle.*
2) *The roots on the unit circle are all simple.*

Definition 6 (Convergence [51])**.** *The linear multi-step method Eq.* (18) *is said to be convergent if all initial value problems satisfy*

$$\lim_{\Delta t \to 0, t_0 + n\Delta t \to t} \mathbf{x}_n = \mathbf{x}(t), \quad t \in (t_0, \tau],$$

whenever the starting values satisfy

$$\lim_{\Delta t \to 0} \mathbf{x}_j = \mathbf{x}(t_0), \quad j = 0, 1, \cdots, k-1.$$

One of the most important results for LMM is the famous slogan [51]:

$$convergence = stability + consistency, \tag{21}$$

meaning that stability and consistency is a necessary and sufficient condition for convergence for the standard k-step LMM. The proof is partly provided in Theorem 4. More details of LMM can be found in Appendix A.1 or Chapter 3 in [52].

4.3.2 Mixed LMM: From NODE to OptDNN

We now show how OptDNNs can be viewed as discretizations of NODEs. Different from the Euler's discretization, we propose a new discretization method called mixed LMM as follows. Consider the first-order NODE of Eq. (14), the formula of this method is

$$-\sum_{i=0}^{k+1} \alpha_i^{k+1} \mathbf{x}_i = \sum_{i=0}^{k+1} \Delta t \beta_i^{k+1} \mathbf{u}_i^{k+1}, \quad \alpha_{k+1}^{k+1} = -1, \tag{22}$$

where $\mathbf{x}_j$ and $\mathbf{u}_j$ are defined as in Eq. (18). Note that the mixed LMM calculates $\mathbf{x}_{k+1}$ using the previous k values with variable step sizes, whereas in the standard k-step LMM, it calculates $\mathbf{x}_{n+k}$ with a fixed step size.

For any OptDNN network of Eq. (15), the parameters α_i^k and β_i^k in Eq. (22) are taken as the modulating parameters in the OptDNN. Discretize the NODE of Eq. (14) using the corresponding mixed LMM method, we obtain the form of the OptDNN network

$$\mathbf{z}_{k+1} = \sum_{i=0}^{k} \alpha_i^{k+1} \mathbf{z}_i + \sum_{i=0}^{k+1} \beta_i^{k+1} \left(\Delta t \mathbf{V}_i^{k+1}\right) \sigma(\mathbf{W}_i^{k+1} \mathbf{z}_i + \mathbf{b}_i^{k+1})$$

for $k \leq L-1$. Here $L = \lceil \frac{\tau}{\Delta t} \rceil$ is the number of layers.

Now, we aim to prove the convergence of the mixed LMM. We first present the following proposition, which states a property of algorithm $\mathscr{A}$.

Proposition 2. *Let $\mathscr{A}$ be a convex optimization algorithm that satisfies all the conditions in Theorem 1. Let matrix $\mathbf{P}_k \in \mathbb{R}^{(k+1)\times k}$ be defined as*

$$\mathbf{P}_k = \begin{pmatrix} \alpha_{k-1}^k & \alpha_{k-2}^k & \cdots & \alpha_1^k & \alpha_0^k \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \in \mathbb{R}^{(k+1)\times k}. \tag{23}$$

Then there exists a vector norm $\|\cdot\|_{op}$ and the induced operator norm, such that $\left\|\prod_{k=s}^{s'} \mathbf{P}_k\right\|_{op}$ is uniformly bounded for all $s' > s$.

Condition 1) in Assumption 2 is the 'consistency' condition, and Proposition 2 is the 'stability' condition for the mixed LMM, respectively. One can understand it from a special case that when the mixed LMM of Eq. (22) is a standard k-step LMM method, $\sum_{i=0}^{k} \alpha_i^{k+1} = 1$ corresponds to the condition that $\rho(1) = 0$ in Eq. (20). Also, the boundedness of $\left\|\prod_{k=s}^{s'} \mathbf{P}_k\right\|_{op}$ is sufficient for the stability of the standard k-step method (See Appendix A.1 for details). With the consistency and stability of the mixed LMM, we can prove the convergence. We give the result in Theorem 3.

Theorem 3. *Let $\mathscr{A}$ be an optimization algorithm that satisfies all the conditions in Theorem 1. $\mathscr{A}$ induces a mixed LMM scheme of Eq.* (22)*. Then the mixed LMM is uniformly convergent for any $\mathbf{x}_0$. Specifically, for any ODE of Eq.* (17)*, it satisfies that*

$$\lim_{\Delta t \to 0, t_0 + k\Delta t \to t} \mathbf{x}_k = \mathbf{x}(t), \quad t \in (t_0, \tau],$$

where $\{\mathbf{x}_k\}$ are derived by solving the ODE using the mixed LMM.

Theorem 3 indicates that OptDNNs have at least the same expressive power as NODEs[5]. Since we have the universality of NODEs in Subsection 4.2, we can derive the universality of OptDNNs. For completeness, we give the proof of Theorem 1 in Appendix B.5. Moreover, we give the approximation rate of our OptDNN for approximating Lipschitz-continuous function from $\mathbb{R}^d$ to $\mathbb{R}^m$ when $m < d$ as follows.

Proposition 3. *Let $g \in C([0,1]^d, \mathbb{R}^m)$ be a continuous function with $\|g\|_{C([0,1]^d)} \leq 1$. Assume that g is L_g-Lipschitz where the smoothness is measure in $\|\cdot\|_\infty$. Under the conditions of Theorem 1 and further assume that $m < d$, there exists an OptDNN network in the form of Eq.* (15) *and Eq.* (16) *with ReLU activation function, $\mathcal{O}(\varepsilon^{-d-m+1})$ depth and $\mathcal{O}(\varepsilon^{-d-m+1})$ weight[6], such that the network function $h : \mathbb{R}^d \to \mathbb{R}^m$, $h(\mathbf{z}_0) = \mathbf{z}_{L+1}$ satisfies*

$$\|g - h\|_{L^p([0,1]^d, \mathbb{R}^m)} \leq \varepsilon.$$

We deem Proposition 3 important in 2 folds: 1) Proposition 3 provides the approximation rate of OptDNNs within a width-bounded setting. In previous works, only limited quantitative results have been provided for FCNN or

5. Note that we do not suggest that the expressive power of OptDNNs is governed by that of NODEs. We provide further discussions in Appendix B.7.

6. The weight refers to the number of all nonzero parameters in a neural network.

ResNet [7], [53]. Specially, our result matches the rate in [7] when $m = 1$ and OptDNN in the form of ResNet as a special case. 2) In the proof of Proposition 3, we also solve a remaining problem in [20] about the rate of approximating continuous function using NODEs when $d > 2$ in time horizon. Note that the authors in [20] only give the rate when $d = 1$ for approximating increasing functions.

5 EXPERIMENTS

Our methodology enables us to design networks from general first-order algorithms. A natural question is if the inspired network is competitive to the commonly used networks, such as ResNet. We first evaluate several examples of OptDNN on the nested ring seperation task and function approximation tasks to validate the universality of OptDNNs. Then we conduct experiments on image classification tasks.

Design OptDNNs We design 8 explicit networks of OptDNN that correspond to different first-order optimization algorithms throughout the experiments. For different tasks, the learnable module T in OptDNN is implemented in different ways. The connections of these OptDNN networks with the existing models and their underlying algorithms are shown in Table 4. We also design 3 new implicit OptDNN networks and conduct experiment on them in image classification. Details of the networks are provided in Appendix C.1.

Practical modifications. In real implementations, we consider some *practical modifications* of OptDNN as follows. Specially, we argue that these modifications do not contradict to our theoretical analysis above (See Appendix C.2).

1) The full connection in T in Eq. (5) can be transformed into convolution. T can also include normalization, downsampling, upsampling, or attention layers.
2) The learnable modules T in Eq. (4) can be of more forms other than in Eq. (5). For example, we can take T as the commonly used 'full pre-activation' function [24]

$$\tilde{T}_i^k(\mathbf{z}) = \mathbf{V}_i^k \sigma_R(\mathbf{W}_i^k \sigma_R(\mathbf{z}) + \mathbf{b}_{i,1}^k) + \mathbf{b}_{i,2}^k, \qquad (24)$$

   where we omit the normalization layers in Eq. (24).
3) The network can be divided into different stages, where the formulations in OptDNN are used in each stage and the size of feature maps may change between the stages.
4) The modulating coefficients in the network can also be set learnable, which is similar to the adaptive step length in optimization algorithms. With this modification, Assumption 2 can be further relaxed.

These modifications are common in optimization-based methods and we will adopt some of them in our experiments. More details are provided in Appendix C.2.

5.1 Nested Ring Separation

We first conduct experiments on a challenging synthetic classification task using our OptDNNs to validate their representation capabilities. It is known that NODEs failed to separate the nested rings [47]. We follow the settings of [38] to try to break apart the 4 nested rings (2 classes) in $\mathbb{R}^2$ using our OptDNNs. Suppose the range of the radius of the i-th ring is $[r_{i,1}, r_{i,2}]$, the target function can be expressed as $\mathbf{1}_{r_{2,1} \le \|\mathbf{x}\|_2 \le r_{2,2}} + \mathbf{1}_{r_{4,1} \le \|\mathbf{x}\|_2 \le r_{4,2}}$. We randomly sample 2,000 points of each ring for each training batch, and sample 500 points of each ring for testing. For all OptDNNs, T follows the formulation in Eq. (5) with hidden dimension $q = 16$ and the ReLU activation, and the network depth is set as 15. All models are trained considering cross-entropy loss with Adam optimizer under the learning rate 0.001 for 1,500 iterations.

The evolution of test points as the depths increase for 4 OptDNNs is shown in Figure 4(a), and the training losses at different scales are shown in Figure 4(b). The results for all OptDNNs are provided in Appendix C.4. It shows that all OptDNNs can successfully separate nested rings, verifying their capabilities. Moreover, OptDNNs inspired by accelerated underlying algorithms usually demonstrate better and faster separation by giving larger classification margins, as well as lower training and testing errors.

5.2 Function Approximation

We then verify the universality of OptDNNs by approximating different functions that are considered difficult for shallow (i.e., 2-layer) neural networks [55], [56]. In the experiment, we choose the parity function [57] and the Telgarski function [56] as the target function. It has been shown that the parity function cannot be approximated better than trivial classifier up to an inverse polynomial when the hypothesis class is 2-layer NNs with polynomial width and weight magnitude [58], and Telgarski function cannot be approximated using an NN with less than $n^{\frac{1}{3}}$ layers up to a constant accuracy (n is defined in Eq. (25)) [56].

Learning Parity Function. Parity function (over n-bits) considers the input space $\mathcal{X}_n = \{\pm 1\}^n$ and a subset $I \subseteq [n]$, with the target function as $f_I(\mathbf{x}) = \prod_{i \in I} x_i$. Note that learning parities can be formulated as a binary classification problem. We consider $n = 16$, formulating a dataset with 2^{16} samples, and randomly sample I with $|I| = 12$ to generate labels. For all OptDNNs, T follows the formulation in Eq. (5) with hidden dimension $q = 64$, and the ReLU activation, and the network depth is set as 15, with another final linear classification layer. All models are trained considering cross-entropy loss with AdamW optimizer under the learning rate 0.001, weight decay 2e-5, and cosine annealing scheduler for 3,000 iterations, with batch size 2,000.

As shown in Figure 5(a), all OptDNNs can be optimized to nearly zero loss, with 100% classification accuracy, demonstrating their capability to learn parity function.

Regression of Telgarski Function. $\forall n$, Telgarski function has the form:

$$f_n(\mathbf{x}) = \begin{cases} 1 & \exists\, t \in \mathbb{N}, x_1 \in \left[\dfrac{2t}{2^n}, \dfrac{2t+1}{2^n}\right], \\ -1 & \text{otherwise}, \end{cases} \qquad (25)$$

with the input space $\mathcal{X}_n = [0, 1]^d$. We consider the regression problem for this function, with $d = 2, n = 4$. For all OptDNNs, T follows the formulation in Eq. (5) with hidden dimension $q = 128$ and the ReLU activation, and the network depth is set as 50, with a linear layer expanding the dimension of inputs to 128 at first and a linear layer

TABLE 4
The connections of the eight OptDNN networks with the existing structures and their underlying optimization algorithms. We say that an OptDNN network is similar to another network if its topology, i.e., the connections between layers, are the same by changing the modulating parameters in the OptDNN network. OptDNN-AGD1 and OptDNN-AGD2 are inspired by two different forms of the accelerated gradient method. OptDNN-APG1, OptDNN-APG2 & OptDNN-APG3 and OptDNN-LADMM1 & OptDNN-LADMM2 are inspired by different kinds of accelerated proximal gradient methods and linearized ADMM, respectively. The details of the underlying algorithms can be found in Appendix C.

Model	Similar Existing models	Underlying Algorithms
OptDNN-HB	HB-NN [18] & MResNet [38]	Heavy Ball Algorithm
OptDNN-AGD1	AGD-NN1 [18]	Accelerated Gradient Descent
OptDNN-AGD2	DenseNet [11]	Accelerated Gradient Descent
OptDNN-APG1	AGD-NN2 [18]	Accelerated Proximal Gradient 1
OptDNN-APG2	N.A.	Accelerated Proximal Gradient 2
OptDNN-APG3	N.A.	Accelerated Proximal Gradient 3
OptDNN-LADMM1	N.A.	Linearized ADMM 1
OptDNN-LADMM2	DMRNet [54]	Linearized ADMM 2

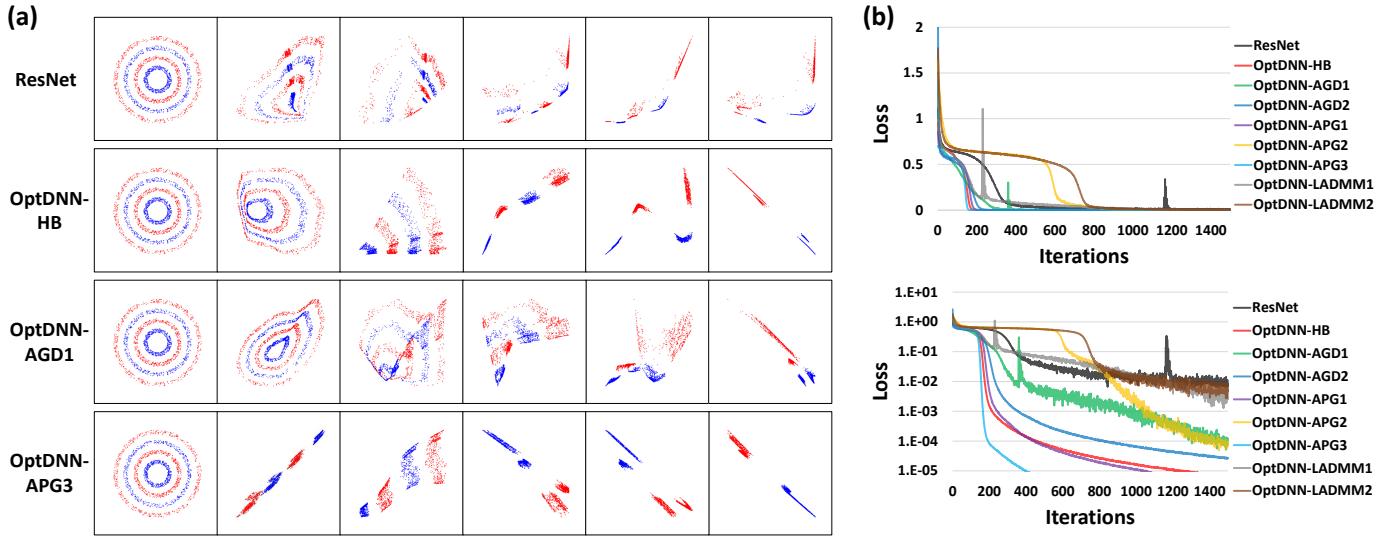


Fig. 4. Results of nested ring separation. (a) Evolution of data points as the depths of networks increase. From left to right, each figure represents data points transformed at layer $3k$. (b) Training losses of different networks at normal and log scale.

transforming the output dimension to 1 at last. All models are trained considering mean square error loss with AdamW optimizer under the learning rate 0.001, weight decay 0.02, and cosine annealing scheduler for 50,000 iterations. For each training iteration, we randomly sample a batch of 50,000 points, and we test models over grid points on $[0, 1]^2$ with an interval of 0.001.

Figure 5(b) shows the training losses of different models, all of which can achieve nearly zero loss. Figure 5(c) presents the test losses of different models and Figure 5(d) visualizes the function realized by an OptDNN over grid points. The results demonstrate that OptDNNs can successfully approximate the difficult Telgarski function, verifying the universality[7].

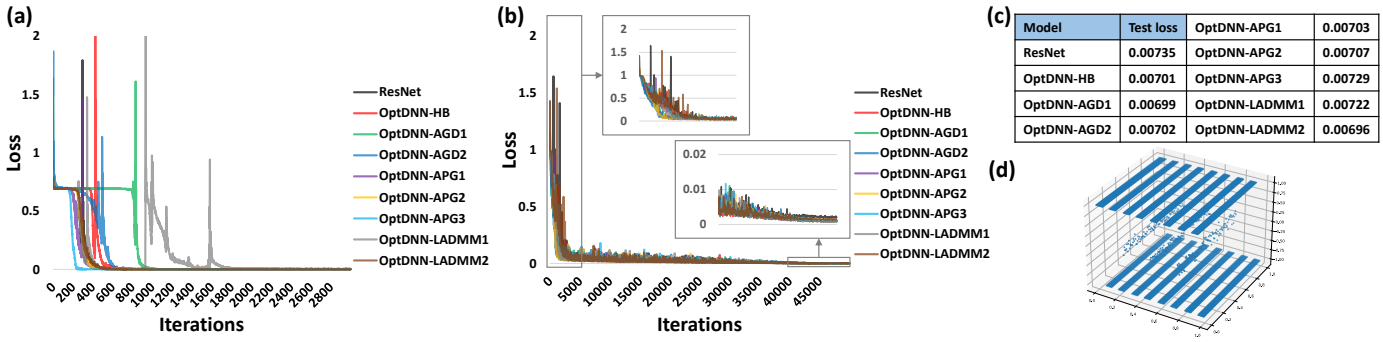
5.3 Image Classification

We further perform experiments on image classification of real datasets.

Datasets. We perform experiments on the commonly used CIFAR-10 and CIFAR-100 datasets [59] as well as the ImageNet dataset [60]. Both CIFAR-10 and CIFAR-100 datasets consist of 60,000 color images with 32×32 pixels, where 50,000 are training samples and 10,000 are testing samples. The CIFAR-10 dataset has 10 classes of objects while the CIFAR-100 dataset has 100 classes of objects. ImageNet-1K is a large-scale dataset of color images with 1,000 classes of objects, which contains 1,281,167 training images and 50,000 validation images. For CIFAR-10 and CIFAR-100, we follow the common practice to normalize the images by subtracting the means and dividing by the standard deviation and applying random cropping, horizontal flipping, and cutout [61] for data augmentation. For comparison experiments with implicit models, we follow their settings and do not use cutout. For ImageNet, we follow the advanced preprocessing and data augmentation setting in [27] and [28].

Architectures. The architectures of our networks consist of multiple blocks corresponding to the iterations of the optimization algorithm. By default, following common practice, each T is implemented as BN-ReLU-Conv-BN-ReLU-Conv, which is written as Eq. (24). T can also take formulation as modules in advanced architectures such as ConvNeXt [27] and ViT [28], which will be specified in experiments. For implicit models, we follow the common

7. One may notice that there are a few inconsistent data points in Figure 5(d). This does not contradict to the universality of OptDNN, since the target function is discontinuous and the universality is proved in the sense of L^p-norm.



Model	Test loss	OptDNN-APG1	0.00703
ResNet	0.00735	OptDNN-APG2	0.00707
OptDNN-HB	0.00701	OptDNN-APG3	0.00729
OptDNN-AGD1	0.00699	OptDNN-LADMM1	0.00722
OptDNN-AGD2	0.00702	OptDNN-LADMM2	0.00696

Fig. 5. Results of function approximation. (a) Training losses (cross-entropy) of different networks on parity function approximation. (b) Training losses (mean square error) of different networks on Telgarski function regression. (c) Test losses (mean square error) of different networks on Telgarski function. (d) Visualization of Telgarski function approximated by OptDNN.

TABLE 5
Results on CIFAR-10 and CIFAR-100 in the shallower and wider setting.

Model	CIFAR-10			CIFAR-100		
	Params	FLOPs	Error (%)	Params	FLOPs	Error (%)
ResNet (GD) [24]	10.67M	0.95G	4.33±0.08	10.72M	0.95G	21.88±0.16
OptDNN-HB	10.67M	0.95G	4.80±0.11	10.72M	0.95G	22.02±0.14
OptDNN-AGD1	10.67M	0.95G	4.37±0.06	10.72M	0.95G	22.12±0.10
OptDNN-AGD2	10.80M	0.86G	**4.27±0.12**	10.84M	0.86G	**21.35±0.18**
OptDNN-APG1	10.80M	0.86G	**4.19±0.13**	10.84M	0.86G	**21.21±0.29**
OptDNN-APG2	10.67M	0.95G	**3.93±0.08**	10.72M	0.95G	**21.36±0.14**
OptDNN-APG3	10.80M	0.86G	**4.12±0.12**	10.84M	0.86G	**21.26±0.44**
OptDNN-LADMM1	10.67M	0.95G	**4.06±0.03**	10.72M	0.95G	**21.45±0.11**
OptDNN-LADMM2	10.67M	0.95G	5.62±0.23	10.72M	0.95G	23.44±0.34

TABLE 6
Results on CIFAR-10 and CIFAR-100 in the deeper and narrower setting.

Model	CIFAR-10			CIFAR-100		
	Params	FLOPs	Error (%)	Params	FLOPs	Error (%)
ResNet (GD) [24]	1.74M	0.53G	4.34±0.14	1.74M	0.53G	22.31±0.20
OptDNN-HB	1.74M	0.53G	4.73±0.08	1.74M	0.53G	**22.28±0.22**
OptDNN-AGD1	1.74M	0.53G	4.39±0.16	1.74M	0.53G	**22.12±0.25**
OptDNN-AGD2	1.73M	0.58G	4.63±0.08	1.74M	0.58G	23.11±0.10
OptDNN-APG1	1.73M	0.58G	**4.32±0.17**	1.74M	0.58G	**22.09±0.26**
OptDNN-APG2	1.74M	0.53G	4.67±0.20	1.74M	0.53G	**21.91±0.18**
OptDNN-APG3	1.73M	0.58G	**4.08±0.02**	1.74M	0.58G	**22.30±0.28**
OptDNN-LADMM1	1.74M	0.53G	4.37±0.11	1.74M	0.53G	22.66±0.08
OptDNN-LADMM2	1.74M	0.53G	4.67±0.08	1.74M	0.53G	22.83±0.36

practice to replace BN with group normalization (GN). We will first consider the 8 explicit networks with default module T in Table 4, and we consider ResNet as a baseline that corresponds to the plain gradient descent algorithm. We also give a demonstration of the architectures of OptDNN-APG2 and OptDNN-APG3[8] in Figure 6. OptDNN-APG2 can be represented as

$$\begin{aligned}\mathbf{y}_k &= \theta_k \mathbf{z}_k + (1-\theta_k)\mathbf{x}_k,\\ \mathbf{z}_{k+1} &= \frac{2}{1+\theta_k}\mathbf{y}_k + \frac{\theta_k - 1}{1+\theta_k}\mathbf{x}_k + T^k(\mathbf{y}_k),\\ \mathbf{x}_{k+1} &= (1-\theta_k)x_k + \theta_k z_{k+1}.\end{aligned}$$

8. It will be shown that these two networks give better performance in our experiment in the rest of this subsection.

And OptDNN-APG3 can be represented as

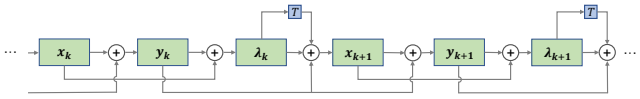
$$\begin{aligned}\mathbf{y}_k =& \theta_k \mathbf{z}_k + (1-\theta_k)\mathbf{x}_k,\\ \mathbf{z}_{k+1} =& \frac{\mathbf{x}_0}{1+\sum_{i=0}^k \theta_i} - \sum_{i=0}^k \frac{\mathbf{y}_i}{\left(1+\sum_{j=0}^k \theta_j\right)\theta_i} + \sum_{i=0}^k T_i^k(\mathbf{y}_i),\\ \mathbf{x}_{k+1} =& (1-\theta_k)\mathbf{x}_k + \theta_k \mathbf{z}_{k+1}.\end{aligned}$$

Besides, to demonstrate the compatibility with existing works, we will show that OptDNNs can also be implemented in similar settings as ResNet, DenseNet, or ConvNeXt. This shows the broad range of OptDNNs either with new strong architectures or incorporating existing design experiences. Detailed descriptions of the architectures are provided in Appendix C.1.

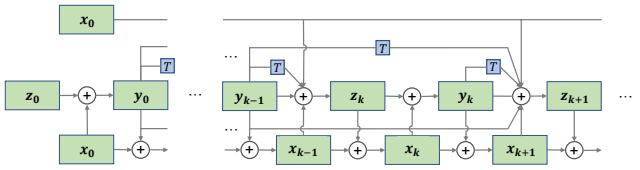
Concretely, on CIFAR-10 and CIFAR-100, we conduct experiments for the 8 explicit OptDNN networks. To comprehensively evaluate the architectures, we consider two settings, i.e. the shallower and wider setting similar to the commonly used ResNet-18, as well as the deeper and

TABLE 7
Results on CIFAR-10 and CIFAR-100 in similar settings as ResNet and DenseNet.

Model	CIFAR-10			CIFAR-100		
	Params	FLOPs	Error (%)	Params	FLOPs	Error (%)
PreActResNet-110 [24]	1.73M	0.26G	4.17	1.74M	0.26G	23.22
OptDNN-HB	1.74M	0.26G	4.59	1.74M	0.26G	**22.72**
OptDNN-AGD1	1.74M	0.26G	4.41	1.74M	0.26G	**22.95**
OptDNN-APG2	1.74M	0.26G	**4.11**	1.74M	0.26G	**22.27**
OptDNN-LADMM1	1.73M	0.26G	4.38	1.74M	0.26G	23.49
OptDNN-LADMM2	1.74M	0.26G	4.39	1.75M	0.26G	23.67
DenseNet-C-52 [11]	1.05M	0.37G	4.05	1.08M	0.37G	23.41
OptDNN-AGD2	1.05M	0.37G	4.32	1.08M	0.37G	**23.30**
OptDNN-APG1	1.17M	0.38G	4.51	1.20M	0.38G	**23.14**
OptDNN-APG3	1.17M	0.38G	4.48	1.21M	0.38G	**22.89**



(a) The architecture of OptDNN-APG2



(b) The architecture of OptDNN-APG3

Fig. 6. The structures of OptDNN-APG2 (a) and OptDNN-APG3 (b). Both networks have three parallel paths and we use $\mathbf{z}, \mathbf{x}, \mathbf{y}$ to illustrate them. In OptDNN-APG3, each $\mathbf{z}_{k+1}$ has connections to $\mathbf{x}_0$ and $\mathbf{y}_0, \mathbf{y}_1, \cdots, \mathbf{y}_k$.

TABLE 8
Results on ImageNet.

Model	Params	FLOPs	Top-1 error (%)
EfficientNet-B4 [62]	19M	4.2G	17.1
RegNetY-8GF [63]	39M	8.0G	18.3
DeiT-S [28]	22M	4.6G	20.2
Swin-T [27]	28M	4.5G	18.7
ConvNeXt-T [27]	29M	4.5G	17.9
OptDNN-APG2	29M	4.5G	17.8

TABLE 9
Results on ImageNet with isotropic structures.

Model	Params	FLOPs	Top-1 error (%)
ConvNeXt-S (iso.) [27]	22M	4.3G	20.3
OptDNN-APG2 (iso.)	22M	4.3G	19.8
ViT-S [28]	22M	4.6G	20.2
OptDNN-APG2-ViT	22M	4.6G	19.6

narrower setting similar to the commonly used ResNet-110. Based on the results on CIFAR, we choose OptDNN-APG2 with overall good performance for experiments on ImageNet.

Training Details. For CIFAR-10 and CIFAR-100, all the models are trained by stochastic gradient descent (SGD) with 0.9 Nesterov momentum and 5×10^{-4} weight decay for 300 epochs. We adopt the weight initialization method in [67] for convolutional layers and use Xavier initialization in [68] for the fully connected layer. We set the batch size set as 128 and use a cosine annealing learning rate scheduler with 0.1 as the initial learning rate and 300 as the number of epochs since the last restart. We set the same random seed for all the experiments. For ImageNet, we follow the same advanced training setting as ConvNeXt [27] or ViT [28].

Results. We first perform experiments on CIFAR. The experiment results with a shallower and wider setting are shown in Table 5, and the results with a deeper and narrower setting are shown in Table 6. We highlight the better results of OptDNNs in boldface. The dynamics of the training accuracy and test accuracy are shown in Appendix C.3. We also perform experiments on CIFAR under similar settings as PreActResNet and DenseNet and present the results in Table 7. Note that our framework is not confined to specific learnable modules. Furthermore, it provide guidance on designing the connections among different modules in DNNs, differing from the module design, i.e., methods that focus on designing the structures of individual modules in most previous works. Thus, our approach complements existing methods and can be applied in conjunction with modules in most well-established architectures, including ConvNeXt.

On CIFAR-100, our OptDNNs give competitive results in three different settings. Some architectures like OptDNN-APG2 can surpass the baseline by around 0.4%-1% with comparable parameters. It can be found in [36] that under the setting with data augmentation, competitive models such as MobileNet [69], FractalNet [70], and HB-NN [18] only surpass the PreAct-ResNet by less than 0.5% or behave worse, while our model can provide a larger improvement. As for the results on CIFAR-10, since classifying CIFAR-10 is a relatively simple task, with strong training settings, most architectures (except for OptDNN-LADMM2 in a shallower and wider setting) achieve similar performance with differences of less than 0.5%. Furthermore, the FLOPs of the OptDNNs are approximately the same as those of their respective baseline ResNet or DenseNet models. Overall, optimization-inspired networks can all achieve competitive performance under different settings and may perform better on more difficult tasks such as CIFAR-100.

We also experiment on ImageNet to compare OptDNNs with the state-of-the-art CNN, ConvNeXt [27], and we consider their tiny model ConvNeXt-T. Our concern revolves around the ability of OptDNNs to achieve comparable results under advanced settings, encompassing the block structure, hyper-parameters, optimizer selection, and other

TABLE 10
Comparison results with implicit models under similar settings on CIFAR-10 and CIFAR-100. Details of the implicit OptDNN networks are provided in Appendix C. Our results are reported as Mean±Std (Max).

Model	CIFAR-10		CIFAR-100	
	Params	Accuracy (%)	Params	Accuracy (%)
ResNet-18 (explicit) [64]	10M	92.9±0.2	/	/
MonDEQ [65]	1M	89.7	/	/
MDEQ [64], [66]	10M	93.8±0.3	11M	72.4±0.2
MOptEqs [66]	8.1M	94.6	8.1M	74.7
ImpOptDNN-PGD	10.67M	93.79±0.34 (94.10)	10.72M	74.24±0.08 (74.33)
ImpOptDNN-FISTA	10.67M	93.10±0.80 (94.14)	10.72M	74.14±0.38 (74.64)
ImpOptDNN-APG	10.67M	94.13±0.06 (94.19)	10.72M	74.60±0.23 (74.88)

TABLE 11
Results of OptDNN-HB with different settings on the parameter β.

Setting	Error (%)	
	CIFAR-10	CIFAR-100
learnable	4.73±0.08	22.28±0.22
fixed as $\frac{1}{3}$	4.38±0.14	22.88±0.18
fixed as 1	5.54±0.04	26.23±0.54

factors. We implement OptDNN-APG2 in a similar setting as ConvNeXt-T and follow the same training settings. As shown in Table 8, with the same amount of parameters and FLOPs, OptDNN-APG2 demonstrates slightly higher performance than the state-of-the-art ConvNeXt-T, showing the competitive performance of optimization-inspired networks. Furthermore, we explore if there are specific scenarios under which OptDNN can offer additional benefits over ConvNext. We hypothesize that OptDNN-APG2 will excel when a Vision Transformer (ViT) style isotropic structure is required, i.e., there is no downsampling layers and the feature resolutions are the same at all depths. This is because isotropic structure may better align with the iterative optimization process compared to hierarchical models with transition modules. To test our hypothesis, we compare OptDNN-APG2 with ConvNext-S in the ViT-style isotropic setting. As shown in Table 9, OptDNN-APG2 (iso.) outperforms ConvNeXt-S (iso.) by 0.5% under similar settings and FLOPs, which validates our hypothesis and highlights the advantages of OptDNNs to an extent.

As introduced previously, our framework can also design implicit models. We design three new OptDNN networks using proximal gradient descent, FISTA, and accelerated proximal gradient descent, respectively. We call them ImpOptDNN to address that they are implicit networks. We compare our ImpOptDNNs with existing implicit models: MDEQ [64], MonDEQ [65], and MOptEqs [66] under similar settings. As shown in Table 10, ImpOptDNN-PGD, ImpOptDNN-FISTA, and ImpOptDNN-APG can achieve competitive results on both CIFAR-10 and CIFAR-100. The results illustrate the potential of our optimization-inspired framework for implicit models as well.

Effect of modulating coefficients. From the proof of Theorem 3, the modulating coefficients can affect the stability (See Definition 5) of the approximation, which is also important for numerical stability. To experimentally show the effect of the modulating coefficients, we train the OptDNN-HB with different settings of the coefficient β on CIFAR in the deeper and narrower setting. We consider that β is fixed as 1, which is not stable, and β is fixed as $\frac{1}{3}$, which is stable[9]. We also consider learnable β initialized as 1. The result is shown in Table 11. The stable $\beta = \frac{1}{3}$ surpass the unstable $\beta = 1$ by 1.5% on CIFAR-10 and 3.4% on CIFAR-100. The learnable β can break the 'instability' and help improve the accuracy. These results accord with our theoretical analysis and also show the necessity of Condition 3) in Assumption 2.

Extension to Transformers [71]. Considering the current trends that Transformer is widely used in computer vision [72], [73], we also make an attempt to extend our framework to design Transformer-style models. A ViT block can be expressed as

$$z = x + \text{Attn}(x) + \text{MLP}(x + \text{Attn}(x)),$$

where $\text{Attn}(\cdot)$ refers to the attention layer and $\text{MLP}(\cdot)$ refers to the MLP layer in ViT block. We derive OptDNN-ViT by defining the learnable module T to be $(\text{Attn}(x) + \text{MLP}(x + \text{Attn}(x)))$. We then conduct experiments on ImageNet under the same setting as ViT-S in [28] and present the result in Table 9. Our OptDNN-APG2-ViT surpassed ViT-S and ConvNext-S (iso.) by approximately 0.6% and 0.7%, respectively, indicating the possibility and potential of our framework to design Transformer-style architectures or to evolve by integrating aspects of the ViT architecture.

In summary, the experiments demonstrate that one can design new strong networks using first-order algorithms. This indicates the reliability of our OptDNN framework.

6 Conclusion and Future Work

Designing the architecture of neural networks is a core research topic in deep learning. This paper proposes a unified framework to design DNNs that have universal approximation guarantees. The framework generalizes the existing optimization-based design method and shows that any convergent first-order algorithms are capable of inspiring new architectures with universality under mild conditions. Finally, several new architectures are developed using our framework, which are shown to be very competitive from empirical study. Our work brings new insights into approximation theory by bridging connections between NODEs and neural networks with general skip connections.

9. When $\beta = 1$, the corresponding 2-step LMM does not satisfy the root condition in Definition 5, whereas, the LMM of $\beta = \frac{1}{3}$ satisfies the root condition. Moreover, $\beta = \frac{1}{3}$ corresponds to the classical 2-step Gear method [52].

We would say that the universal approximation property should be the basic requirement on learning models to work on unknown complex real tasks. Our framework paves the way to obtain architectures with such a guarantee. There are lots of directions that remain to study based on our framework. First, this work only studies first-order algorithms, so it is interesting to study high-order algorithms such as the Newton method [36]. Second, since our framework can also understood from a first-order ODE scheme perspective, it is intriguing to explore the design of networks based on higher-order ODE schemes. Third, it is expecting to design novel modules that are tailored for OptDNNs. Moreover, we expect to use our view that links the architecture to an algorithm to understand some practical tricks and adaptations, such as the attention mechanism [71], in neural networks. Besides, it is also possible to incorporate our framework with other methodologies, such as providing initial architectures for manual design or NAS.

ACKNOWLEDGMENTS

Cong Fang and Zhouchen Lin were supported by National Key R&D Program of China (2022ZD0160302), the NSF China (No. 62276004), and the major key project of PCL, China (No. PCL2021A12).

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of Go without human knowledge," *Nature*, 2017.

[4] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, 1989.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, 1989.

[6] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems*, 2017.

[7] H. Lin and S. Jegelka, "ResNet with one-neuron hidden layers is a universal approximator," in *Advances in Neural Information Processing Systems*, 2018.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[9] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[12] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[13] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations*, 2017.

[14] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[15] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *International Conference on Machine Learning*, 2010.

[16] Y. Yang, J. Sun, H. Li, and Z. Xu, "ADMM-CSNet: A deep learning approach for image compressive sensing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[17] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Processing Magazine*, 2021.

[18] H. Li, Y. Yang, D. Chen, and Z. Lin, "Optimization algorithm inspired deep neural network structure design," in *Asian Conference on Machine Learning*, 2018.

[19] A. Bibi, B. Ghanem, V. Koltun, and R. Ranftl, "Deep layers as stochastic solvers," in *International Conference on Learning Representations*, 2019.

[20] Q. Li, T. Lin, and Z. Shen, "Deep learning via dynamical systems: An approximation perspective," *Journal of the European Mathematical Society*, 2022.

[21] F. Bashforth and J. C. Adams, *An Attempt to Test the Theories of Capillary Action by Comparing the Theoretical and Measured Forms of Drops of Fluid*. University Press, 1883.

[22] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$," in *Doklady an USSR*, 1983.

[23] N. Parikh, S. Boyd *et al.*, "Proximal algorithms," *Foundations and Trends® in Optimization*, 2014.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*, 2016.

[25] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," in *Advances in Neural Information Processing Systems*, 2019.

[26] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Tsai, "Implicit deep learning," *SIAM Journal on Mathematics of Data Science*, 2021.

[27] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[28] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, 2021.

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

[30] B. Hanin and M. Sellke, "Approximating continuous functions by ReLU nets of minimal width," *arXiv preprint arXiv:1710.11278*, 2017.

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

[32] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," in *Conference on Learning Theory*, 2020.

[33] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," *arXiv preprint arXiv:1409.2574*, 2014.

[34] J. Adler and O. Öktem, "Learned primal-dual reconstruction," *IEEE Transactions on Medical Imaging*, 2018.

[35] K. H. R. Chan, Y. Yu, C. You, H. Qi, J. Wright, and Y. Ma, "Deep networks from the principle of rate reduction," *arXiv preprint arXiv:2010.14765*, 2020.

[36] Z. Shen, Y. Yang, Q. She *et al.*, "Newton design: Designing CNNs with the family of Newton's methods," *Science China Information Science*, 2022.

[37] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, 1993.

[38] M. E. Sander, P. Ablin, M. Blondel, and G. Peyré, "Momentum residual neural networks," in *International Conference on Machine Learning*, 2021.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[40] M. Li, L. He, and Z. Lin, "Implicit Euler skip connections: Enhancing adversarial robustness via numerical stability," in *International Conference on Machine Learning*, 2020.

[41] X. Xie, Q. Wang, Z. Ling, X. Li, G. Liu, and Z. Lin, "Optimization induced equilibrium networks: An explicit optimization perspective for understanding equilibrium models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[42] Y. Lu, A. Zhong, Q. Li, and B. Dong, "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations," in *International Conference on Machine Learning*, 2018.
[43] L. Ruthotto and E. Haber, "Deep neural networks motivated by partial differential equations," *Journal of Mathematical Imaging and Vision*, 2020.
[44] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, 1964.
[45] Z. Lin, R. Liu, and Z. Su, "Linearized alternating direction method with adaptive penalty for low-rank representation," *Advances in Neural Information Processing Systems*, 2011.
[46] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM Journal on Optimization*, 2016.
[47] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural ODEs," *Advances in Neural Information Processing Systems*, 2019.
[48] H. Zhang, X. Gao, J. Unterman, and T. Arodz, "Approximation capabilities of neural ODEs and invertible residual networks," in *International Conference on Machine Learning*, 2020.
[49] T. Teshima, K. Tojo, M. Ikeda, I. Ishikawa, and K. Oono, "Universal approximation property of neural ordinary differential equations," *arXiv preprint arXiv:2012.02414*, 2020.
[50] E. Weinan, "A proposal on machine learning via dynamical systems," *Communications in Mathematics and Statistics*, 2017.
[51] G. Dahlquist, "Convergence and stability in the numerical integration of ordinary differential equations," *Mathematica Scandinavica*, 1956.
[52] G. Wanner and E. Hairer, *Solving Ordinary Differential Equations I*. Springer Berlin Heidelberg New York, 1993.
[53] B. Hanin, "Universal function approximation by deep neural nets with bounded width and ReLU activations," *Mathematics*, 2019.
[54] L. Zhao, J. Wang, X. Li, Z. Tu, and W. Zeng, "Deep convolutional neural networks with merge-and-run mappings," *arXiv preprint arXiv:1611.07718*, 2016.
[55] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," in *Conference on Learning Theory*, 2016.
[56] M. Telgarsky, "Benefits of depth in neural networks," in *Conference on Learning Theory*, 2016.
[57] M. Furst, J. B. Saxe, and M. Sipser, "Parity, circuits, and the polynomial-time hierarchy," *Mathematical Systems Theory*, 1984.
[58] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, "The connection between approximation, depth separation and learnability in neural networks," in *Conference on Learning Theory*, 2021.
[59] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," in *Citeseer, Tech. Rep.*, 2009.
[60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.
[61] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.
[62] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019.
[63] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.
[64] S. Bai, V. Koltun, and J. Z. Kolter, "Multiscale deep equilibrium models," *Advances in Neural Information Processing Systems*, 2020.
[65] E. Winston and J. Z. Kolter, "Monotone operator equilibrium networks," *Advances in Neural Information Processing Systems*, 2020.
[66] M. Li, Y. Wang, X. Xie, and Z. Lin, "Optimization inspired multi-branch equilibrium models," in *International Conference on Learning Representations*, 2021.
[67] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.
[68] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2010.
[69] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[70] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-deep neural networks without residuals," in *International Conference on Learning Representations*, 2017.
[71] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.
[72] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.
[73] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.
[74] T. H. Gronwall, "Note on the derivatives with respect to a parameter of the solutions of a system of differential equations," *Annals of Mathematics*, 1919.
[75] G. Strang, *Linear Algebra and Its Applications*. Belmont, CA: Thomson, Brooks/Cole, 2006.
[76] S. D. Conte and C. De Boor, *Elementary Numerical Analysis: An Algorithmic Approach*. SIAM, 2017.
[77] F. W. Warner, *Foundations of Differentiable Manifolds and Lie Groups*. Springer Science & Business Media, 1983.
[78] Z. Lin, H. Li, and C. Fang, *Accelerated Optimization for Machine Learning - First-Order Algorithms*. Springer, 2020.
[79] A. Beck, *First-order Methods in Optimization*. SIAM, 2017.
[80] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015.
[81] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.
[82] P. Petersen and F. Voigtlaender, "Equivalence of approximation by convolutional neural networks and fully-connected networks," *Proceedings of the American Mathematical Society*, 2020.
[83] K. Oono and T. Suzuki, "Approximation and non-parametric estimation of ResNet-type convolutional neural networks," in *International Conference on Machine Learning*, 2019.
[84] D.-X. Zhou, "Theory of deep convolutional neural networks: Downsampling," *Neural Networks*, 2020.
[85] J. Schmidt-Hieber, "Nonparametric regression using deep neural networks with ReLU activation function," *The Annals of Statistics*, 2020.

Zhoutong Wu received the B.S. degree in information and computing science from Fudan University in 2022. He is currently pursuing the Master degree in Center for Data Science, Academy for Advanced Interdisciplinary Studies, Peking University. His research interests include machine learning and optimization.

Mingqing Xiao is currently pursuing the Ph.D. degree in the School of Intelligence Science and Technology, Peking University. He received the B.S. degree in computer science and technology and the double B.S. degree in psychology from Peking University in 2020. His research interests include machine learning, optimization, and brain-inspired computing.

Cong Fang is now an Assistant Professor with Peking University, Beijing, China. He received the Ph.D. degree from Peking University in 2019 and was a Post-Doctoral Researcher with Princeton University in 2020 and the University of Pennsylvania in 2021. His research interests include machine learning and optimization.

Zhouchen Lin (M'00-SM'08-F'18) received the Ph.D. degree in applied mathematics from Peking University in 2000. He is currently a Boya Special Professor with the National Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University. His research interests include machine learning and numerical optimization. He has published over 290 papers, collecting more than 31,000 Google Scholar citations. He is a Fellow of the IAPR, the IEEE, the AAIA, and the CSIG.

Supplementary Material

APPENDIX A
BACKGROUND

A.1 The Linear Multi-step Method (LMM)

In this section, we give a brief introduction to LMM. The LMM first appeared in the work of [21]. A general theory of LMM was started in [51]. We refer the readers to Chapter 3 in [52] for a comprehensive introduction to LMM.

Recall that the k-step LMM formula for solving the ODE of Eq. (17) can be represented as

$$\sum_{j=0}^{k} \alpha_j \mathbf{x}_{n+j} = \Delta t \sum_{j=0}^{k} \beta_j \mathbf{u}_{n+j}. \tag{26}$$

And the generating polynomials are defined as in Eq. (19). Now we introduce the concept of the local truncation error.

Definition 7 (Local truncation error [51])**.** *The local truncation error of the LMM in Eq.* (26) *is defined as*

$$\mathbf{r}_{n+k} = \sum_{j=0}^{k} [\alpha_j \mathbf{x}(t_n + j\Delta t) - \Delta t \beta_j \dot{\mathbf{x}}(t_n + j\Delta t)].$$

With the definition of local truncation error, the order of LMM can be equally defined as follows.

Definition 8 (Order [51])**.** *The multi-step method Eq.* (26) *is said to be of order p' if the local truncation error $\mathbf{r}_n$ satisfies*

$$\mathbf{r}_{n+k} = \mathbf{c}\Delta t^{p'+1} + \mathcal{O}(\Delta t^{p'+2}),$$

where $\mathbf{c} \in \mathbb{R}^d$ is a non-zero vector. Moreover, the conditions for a multi-step method to be of order 1 are usually called consistency conditions.

The stability and convergence of the LMM are defined in Definition 5 and 6, respectively.

Next, we will briefly prove that stability and consistency are sufficient for convergence, which is known as a part of the famous slogan [51]

$$convergence = stability + consistency.$$

To help better understand our proof of the convergence of the mixed LMM in Appendix B, we will give a modified version of the proof of this proposition. We also need the discrete version of Gronwall-Bellman Inequality for the proof.

Lemma 1 (Gronwall-Bellman Inequality [74])**.** *Let k be a nonnegative integer. If the nonnegative sequence $\{x_n\}$ satisfies*

$$x_n \le \alpha + \beta\Delta t \sum_{j=0}^{n-1} x_j, \quad n = k, k+1, \cdots, \left\lceil \frac{\tau}{\Delta t} \right\rceil,$$

where $\alpha, \beta > 0$, then it holds

$$x_n \le e^{\beta\tau} \left(\alpha + \beta k \Delta t \max_{0 \le j \le k-1} x_j \right), \; n = k, k+1, \cdots, \left\lceil \frac{\tau}{\Delta t} \right\rceil.$$

Theorem 4 (Modified version of Theorem 1 in [51])**.** *If the multi-step method of Eq.* (26) *is stable and consistent, then it is convergent.*

Proof. Denote the error by $\mathbf{e}_n = \mathbf{x}(t_n) - \mathbf{x}_n$, by the definition of the local truncation error, it holds that

$$\begin{aligned}\mathbf{r}_{n+k} &= \sum_{j=0}^{k} [\alpha_j \mathbf{x}(t_n + j\Delta t) - \Delta t \beta_j \dot{\mathbf{x}}(t_n + j\Delta t)] \\ &= \sum_{j=0}^{k} [\alpha_j \mathbf{x}(t_n + j\Delta t) - \Delta t \beta_j \mathbf{u}(t_n + j\Delta t, \mathbf{x}(t_n + j\Delta t))].\end{aligned}$$

From Eq. (26), we have

$$\begin{aligned}\sum_{j=0}^{k} \alpha_j \mathbf{e}_{n+j} &= \sum_{j=0}^{k} \alpha_j (\mathbf{x}(t_{n+j}) - \mathbf{x}_{n+j}) \\ &= \sum_{j=0}^{k} \beta_j \Delta t (\mathbf{u}(t_{n+j}, \mathbf{x}(t_{n+j})) - \mathbf{u}(t_{n+j}, \mathbf{x}_{n+j})) \\ &\quad + \mathbf{r}_{n+k}.\end{aligned} \tag{27}$$

Without loss of generality, we assume that $\alpha_k = 1$. From the definition of order, there exist $p' \ge 2$ and a vector $\mathbf{c}$, such that

$$\mathbf{r}_{n+k} = \mathbf{c}\Delta t^{p'+1} + \mathcal{O}(\Delta t^{p'+2}). \tag{28}$$

Define

$$\mathbf{b}_{n+k} = \sum_{j=0}^{k} \beta_j \Delta t (\mathbf{u}(t_{n+j}, \mathbf{x}(t_{n+j})) - \mathbf{u}(t_{n+j}, \mathbf{x}_{n+j})).$$

Equivalently, we can write Eq. (27) as

$$\mathbf{E}_{n+k} = \mathbf{P}\mathbf{E}_{n+k-1} + \mathbf{B}_{n+k}, \quad \forall n \ge 0, \tag{29}$$

where

$$\begin{aligned}\mathbf{E}_{n+k} &= (\mathbf{e}_{n+k}, \mathbf{e}_{n+k-1}, \cdots, \mathbf{e}_{n+1})^T \in \mathbb{R}^{k \times d}, \\ \mathbf{B}_{n+k} &= (\mathbf{b}_{n+k} + \mathbf{r}_{n+k}, \mathbf{0}, \cdots, \mathbf{0})^T \in \mathbb{R}^{k \times d}, \\ \mathbf{P} &= \begin{pmatrix} -\alpha_{k-1} & -\alpha_{k-2} & \cdots & -\alpha_1 & -\alpha_0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \in \mathbb{R}^{k \times k}.\end{aligned}$$

By recursion, we have

$$\mathbf{E}_{n+k} = \mathbf{P}^{n+1}\mathbf{E}_{k-1} + \sum_{m=k}^{n+k} \mathbf{P}^{n+k-m}\mathbf{B}_m. \tag{30}$$

Since the method is stable, the generating polynomial $\rho(\lambda)$ satisfies the root condition. Note that the characteristic polynomial of $\mathbf{P}$ is exactly $\rho(\lambda)$, thus from the root condition and the theory of Jordan canonical form [75], there exists a constant $c_1 > 0$, such that

$$\|\mathbf{P}^n\|_\infty \le c_1, \quad \forall n \ge 0.$$

Hence from Eq. (30) we get

$$\|\mathbf{E}_{n+k}\|_\infty \le c_1 \|\mathbf{E}_{k-1}\|_\infty + c_1 \sum_{m=k}^{n+k} \|\mathbf{B}_m\|_\infty. \tag{31}$$

By the assumption that $\mathbf{u}(t, \mathbf{x})$ is L_u-Lipschitz with respect to $\mathbf{x}$ (Without loss of generality, assume that it is in the sense of $\|\cdot\|_1$.), it holds

$$\|\mathbf{b}_{n+k}\|_1 \le \sum_{j=0}^{k} \beta_j \Delta t L_u \|\mathbf{e}_{n+j}\|_1.$$

Therefore for matrix $\mathbf{B}_m, m \geq k$, it holds

$$\|\mathbf{B}_m\|_\infty \leq \beta L_u \Delta t \sum_{j=0}^{k} \|\mathbf{e}_{m-k+j}\|_1 + \|\mathbf{r}_m\|_1, \tag{32}$$

where $\beta = \max_{0\leq j\leq k-1} |\beta_j|$ and we use the fact that $\|\mathbf{B}_m\|_\infty = \|\mathbf{b}_m + \mathbf{r}_m\|_1$. Combining it with Eq. (31) we have

$$\begin{aligned}\|\mathbf{E}_{n+k}\|_\infty \leq & c_1\|\mathbf{E}_{k-1}\|_\infty + c_1\beta L_u \Delta t \sum_{m=k}^{n+k}\sum_{j=0}^{k} \|\mathbf{e}_{m-k+j}\|_1 \\ & + c_1 \sum_{m=k}^{n+k} \|\mathbf{r}_m\|_1.\end{aligned} \tag{33}$$

By definition 8, there exists a constant c_2 such that $\|\mathbf{r}_i\|_1 \leq c_2 \Delta t^{p'+1}$ for all i, thus we have

$$\begin{aligned}\|\mathbf{e}_{n+k}\|_1 \leq & \|\mathbf{E}_{n+k}\|_\infty \\ \leq & c_1 \left[\|\mathbf{E}_{k-1}\|_\infty + c_2(n+1)\Delta t^{p'+1}\right] \\ & + c_1\beta L_u \Delta t \sum_{j=0}^{k}\sum_{m=k}^{n+k} \|\mathbf{e}_{m-k+j}\|_1 \\ \leq & c_1(\|\mathbf{E}_{k-1}\|_\infty + c_2\tau\Delta t^{p'}) \\ & + c_1\beta L_u \Delta t(k+1) \sum_{m=0}^{n+k} \|\mathbf{e}_m\|_1,\end{aligned} \tag{34}$$

where we use $n+1 \leq \frac{\tau}{\Delta t}$ in the third equation. When Δt is small enough, plugging the Gronwall-Bellman Inequality into Eq. (34), we derive

$$\begin{aligned}&\|\mathbf{e}_{n+k}\|_1 \\ \leq & c_0 e^{c_0\beta L_u (k+1)\tau}((1+\beta L_u \Delta t(k+1)k)\|\mathbf{E}_{k-1}\|_\infty + c_2\tau\Delta t^{p'}),\end{aligned}$$

where

$$c_0 = \frac{c_1}{1-\beta L_u(k+1)\Delta t c_1}, \quad \beta L_u(k+1)\Delta t c_1 < 1.$$

Thus we finish the proof. □

At the end of the section, we introduce two schemes of the commonly used LMM.

Adams Methods. [52] Integrating Eq. (17) on some interval $[t_n, t_{n+1}]$, it holds

$$\mathbf{x}(t_{n+1}) - \mathbf{x}(t_n) = \int_{t_n}^{t_{n+1}} \mathbf{u}(t,\mathbf{x})\mathrm{d}t. \tag{35}$$

The Adams methods are derived by solving the integration of the right-hand side of Eq. (35) numerically by Newton-Cotes methods [76]. For example, the Adams-Moulton scheme is

$$\mathbf{x}_{n+1} - \mathbf{x}_n = \frac{\Delta t}{12}(5\mathbf{u}_{n+1} + 8\mathbf{u}_n - \mathbf{u}_{n-1}),$$

and the Milne scheme can be represented as

$$\mathbf{x}_{n+2} - \mathbf{x}_n = \frac{\Delta t}{3}(\mathbf{u}_{n+2} + 4\mathbf{u}_{n+1} + \mathbf{u}_n).$$

Gear Methods. [52] The Gear methods are also called the methods based on differentiation (BDF). The theme of Gear methods is to approximate the derivative $\dot{\mathbf{x}}(t)$ with numerical methods. For example, the 2-step Gear method (BDF2) is

$$\frac{3}{2}\mathbf{x}_{n+1} - 2\mathbf{x}_n + \frac{1}{2}\mathbf{x}_{n-1} = \Delta t \mathbf{u}_{n+1},$$

and the 3-step Gear method (BDF3) is

$$\frac{11}{6}\mathbf{x}_{n+1} - 3\mathbf{x}_n + \frac{3}{2}\mathbf{x}_{n-1} - \frac{1}{3}\mathbf{x}_{n-2} = \Delta t \mathbf{u}_{n+1}.$$

APPENDIX B
PROOFS AND DISCUSSIONS

B.1 Proof of Proposition 1

Proof. Since K is a compact set and ϕ is continuous, $\phi(K)$ is a compact set in $\mathbb{R}^m$, thus $\phi(K)$ is bounded and closed. For $\varepsilon > 0$, we can find $\{O_i\}_{i=1}^M$ such that $\phi(K) \subset \cup_{i=1}^M O_i$, $O_i \cap \phi(K) \neq \emptyset$, where O_i is an open set whose diameter is smaller than ε for all i.

By the continuity of ϕ, $\phi^{-1}(O_i)$ is an open set in $\mathbb{R}^d$. Hence, $K \subset \cup_{i=1}^M \phi^{-1}(O_i)$, i.e., $\cup_{i=1}^M \phi^{-1}(O_i)$ is an open covering of K. By the theorem of partition of unity [77], for each i, there exists a function $\tilde{\phi}_i \in C(K,[0,1])$, such that

$$\sum_{i=1}^{M} \tilde{\phi}_i(\mathbf{x}) = 1, \forall \mathbf{x} \in K, \quad \operatorname{supp}\tilde{\phi}_i \subset \phi^{-1}(O_i),\ i \in [M].$$

For each $l^{-1}(O_i)$, since l is a surjection, we can randomly choose a $\mathbf{z}_i \in l^{-1}(O_i)$. Define ψ as

$$\psi(\mathbf{x}) = \sum_{i=1}^{M} \tilde{\phi}_i(\mathbf{x})\mathbf{z}_i.$$

We see that ψ is a continuous vector-valued function from K to $\mathbb{R}^d$. Furthermore, since $\phi(K) \subset \cup_{i=1}^M O_i$, for each $\mathbf{x} \in K$, we can find an index j such that $\mathbf{x} \in K \cap \phi^{-1}(O_j)$. Hence it holds

$$\begin{aligned}\phi(\mathbf{x}) - l\circ\psi(\mathbf{x}) = & \phi(\mathbf{x}) - l\left(\sum_{i=1}^{M}\tilde{\phi}_i(\mathbf{x})\mathbf{z}_i\right) \\ = & \phi(\mathbf{x}) - \sum_{i=1}^{M}\tilde{\phi}_i(\mathbf{x})l(\mathbf{z}_i) \\ = & \sum_{i=1}^{M}\tilde{\phi}_i(\mathbf{x})(\phi(\mathbf{x}) - l(\mathbf{z}_i)),\end{aligned} \tag{36}$$

where the second equality holds because l is a linear map, and the third equality holds because $\sum_{i=1}^M \tilde{\phi}_i = 1$. Now we define

$$\begin{aligned}s_1(\mathbf{x}) = & \sum_{i:\mathbf{z}_i \in l^{-1}(O_j)} \tilde{\phi}_i(\mathbf{x})\left(\phi(\mathbf{x}) - l(\mathbf{z}_i)\right), \\ s_2(\mathbf{x}) = & \sum_{i:\mathbf{z}_i \notin l^{-1}(O_j)} \tilde{\phi}_i(\mathbf{x})\left(\phi(\mathbf{x}) - l(\mathbf{z}_i)\right),\end{aligned}$$

Note that $0 \leq \tilde{\phi}_i \leq 1$ for all i. By the fact that $\mathbf{x} \in \phi^{-1}(O_j)$ and the diameter of O_i is smaller than ε, we have

$$\begin{aligned}\|s_1(\mathbf{x})\| \leq & \sum_{i:\mathbf{z}_i \in l^{-1}(O_j)} \tilde{\phi}_i(\mathbf{x}) \cdot \|\phi(\mathbf{x}) - l(\mathbf{z}_i)\| \\ \leq & \sum_{i:\mathbf{z}_i \in l^{-1}(O_j)} \tilde{\phi}_i(\mathbf{x}) \cdot \varepsilon\end{aligned}$$

$$\leq \sum_{i=1}^{M} \tilde{\phi}_i(\mathbf{x}) \cdot \varepsilon \leq \varepsilon. \tag{37}$$

When $\mathbf{z}_i \notin l^{-1}(O_j)$ for some i, since $\tilde{\phi}_i$ is supported at $\phi^{-1}(O_i)$ and $\mathbf{x} \in \phi^{-1}(O_j)$, it holds that $\tilde{\phi}_i(\mathbf{x}) \neq 0$ if and only if

$$\mathbf{x} \in \phi^{-1}(O_j) \cap \phi^{-1}(O_i), \quad O_i \cap O_j \neq \emptyset. \tag{38}$$

Define $I = \{i : \mathbf{z}_i \notin l^{-1}(O_j), O_i \cap O_j \neq \emptyset\}$. For $s_2(\mathbf{x})$, we have

$$\begin{aligned} \|s_2(\mathbf{x})\| &\leq \sum_{i:\mathbf{z}_i \notin l^{-1}(O_j)} \tilde{\phi}_i(\mathbf{x}) \cdot \|\phi(\mathbf{x}) - l(\mathbf{z}_i)\| \\ &\leq \sum_{i \in I} \tilde{\phi}_i(\mathbf{x}) \cdot \|\phi(\mathbf{x}) - l(\mathbf{z}_i)\| \quad (39) \\ &\leq \sum_{i \in I} \tilde{\phi}_i(\mathbf{x}) \cdot 2\varepsilon \quad (40) \\ &\leq 2\varepsilon, \end{aligned}$$

where Eq. (39) holds because of Eq. (38), and Eq. (40) holds because $\phi(\mathbf{x}) \in O_j$, $l(\mathbf{z}_i) \in O_i$ and $O_i \cap O_j \neq \emptyset$ when $i \in I$. Combining all the above results, we have

$$\|\phi(\mathbf{x}) - l \circ \psi(\mathbf{x})\| \leq \|s_1(\mathbf{x})\| + \|s_2(\mathbf{x})\| \leq 3\varepsilon,$$

which finishes the proof.

□

B.2 Proof of Theorem 2

Our proof is based on Theorem 2.3 and Proposition 3.11 in [20]. We first introduce some definitions as follows.

Definition 9 (Attainable set [20])**.** *For a control family $\mathcal{F}$ and a function $\mathbf{u} \in \mathcal{F}$, denote by $\varphi_\tau^{\mathbf{u}}(\mathbf{x}_0)$ the flow map of the following ODE at time τ:*

$$\varphi_\tau^{\mathbf{u}}(\mathbf{x}_0) = \mathbf{x}(\tau; \mathbf{x}_0), \text{ where } \dot{\mathbf{x}}(t) = \mathbf{u}(\mathbf{x}(t)), \ \mathbf{x}(0) = \mathbf{x}_0, \tag{41}$$

where $\mathbf{x}(\tau; \mathbf{x}_0)$ is the value of $\mathbf{x}(\tau)$ when the initial point is $\mathbf{x}_0$. The attainable set of ODE Eq. (41) at time τ is defined as

$$\begin{aligned} \mathcal{A}_{\mathcal{F}}(\tau) = \{&\mathbf{x}_0 \mapsto \varphi_{\tau_k}^{\mathbf{u}_k} \circ \varphi_{\tau_{k-1}}^{\mathbf{u}_{k-1}} \circ \cdots \circ \varphi_{\tau_1}^{\mathbf{u}_1}(\mathbf{x}_0) : \\ &\tau_1 + \cdots + \tau_k = \tau, \quad \mathbf{u}_1, \cdots, \mathbf{u}_k \in \mathcal{F}, \quad k \geq 1\}. \end{aligned}$$

The attainable set at any time is defined as $\mathcal{A}_{\mathcal{F}} = \bigcup_{\tau > 0} \mathcal{A}_{\mathcal{F}}(\tau)$.

Consider the control family $\mathcal{F}_\sigma$:

$$\mathcal{F}_\sigma = \{\mathbf{z} \mapsto \mathbf{V}\sigma(\mathbf{W}\mathbf{z} + \mathbf{b}) : \mathbf{V} \in \mathbb{R}^{d \times q}, \mathbf{W} \in \mathbb{R}^{q \times d}, \mathbf{b} \in \mathbb{R}^q\}. \tag{42}$$

To prove the universality of the NODE

$$\dot{\mathbf{z}}(t) = \mathbf{u}(t, \mathbf{z}), \quad \mathbf{z}(0) = \mathbf{z}_0,$$

where $\mathbf{z} \mapsto \mathbf{u}(t, \mathbf{z}) \in \mathcal{F}_\sigma$, inspired by [20], we also focus on a subset of the first-order NODEs. Specifically, we consider the piece-wise autonomous NODEs

$$\dot{\mathbf{z}}(t) = \mathbf{u}(t, \mathbf{z}), \quad t \in [0, \tau], \quad u(t, \cdot) \in \mathcal{F}_\sigma, \tag{43}$$

where

$$\mathbf{u}(t, \cdot) = \mathbf{u}(t_i, \cdot), \quad t \in [t_i, t_{i+1}) \subset [0, \tau]$$

for $0 = t_1 < t_2 < \cdots < t_N = \tau$. The function class of the piece-wise ODE of Eq. (43) is exactly the attainable set $\mathcal{A}_{\mathcal{F}_\sigma}(\tau)$ of the ODE of Eq. (41).

We need two Propositions from [20] for our proof and we state them as follows.

Proposition 4 (A version of Theorem 2.3 in [20])**.** *Let $d \geq 2, 1 \leq p < \infty$, $g : [0,1]^d \to \mathbb{R}^d$ be a continuous vector-valued function, and $\mathcal{A}_{\mathcal{F}_{\sigma_R}}$ be the attainable set of the ODE of Eq. (41) with the control family of Eq. (42) where $\sigma = \sigma_R$. Then for any $\varepsilon > 0$, there exists an $h \in \mathcal{A}_{\mathcal{F}_{\sigma_R}}$, such that*

$$\|g - h\|_{L^p([0,1]^d, \mathbb{R}^d)} \leq \varepsilon.$$

Proposition 5 (Proposition 3.11 in [20])**.** *Let $\mathcal{F}$ be a Lipschitz control family. Then for any Lipschitz control family $\tilde{\mathcal{F}}$ such that $\mathcal{F} \subset \tilde{\mathcal{F}} \subset \overline{\mathrm{CH}}(\mathcal{F})$, we have*

$$\overline{\mathcal{A}_{\mathcal{F}}} = \overline{\mathcal{A}_{\tilde{\mathcal{F}}}},$$

where $\overline{\mathrm{CH}}(\mathcal{F})$ refers to the closure of the convex hull of $\mathcal{F}$ and $\overline{\mathcal{A}_{\mathcal{F}}}$ refers to the closure of $\mathcal{A}_{\mathcal{F}}$.

Note that in Theorem 2.3 in [20], the authors assume that the dimension q is greater than d. This condition can be relaxed to $q \geq 2$ following their proof technique. For simplicity, we omit the details.

Now we begin our proof of Theorem 2.

Proof. Let $\mathcal{F}_{\sigma_R}$ be the control family defined as in Eq. (42) when $\sigma = \sigma_R$. It can be easily verified that $\mathcal{F}_{\sigma_R}$ is a Lipschitz control family. For any activation function σ that satisfies Assumption 1, we want to show that $\mathcal{F}_\sigma \subset \overline{\mathrm{CH}}(\mathcal{F}_{\sigma_R})$. Then by Proposition 4 and 5, we will have

$$\overline{\mathcal{A}_{\mathcal{F}_\sigma}} = \overline{\mathcal{A}_{\mathcal{F}_{\sigma_R}}}.$$

Since $\mathcal{A}_{\mathcal{F}_{\sigma_R}}$ is dense in the sense of $\|\cdot\|_{L^p([0,1]^d, \mathbb{R}^d)}$ for functions in $C([0,1]^d, \mathbb{R}^d)$ by Proposition 4, we can derive the L^p-universality of $\mathcal{A}_{\mathcal{F}_{\sigma_R}}$ and so of NODEs.

For any $\varepsilon > 0$, suppose that there exists a function

$$r : \mathbb{R} \to \mathbb{R}, \quad r(x) = \sum_{i=1}^{s} m_i \sigma_R(n_i x + p_i), \tag{44}$$

where $s \geq 1$, m_i, n_i, p_i are some scalars, such that

$$\|\sigma - r\|_{L^\infty(\mathbb{R})} \leq \varepsilon.$$

Then for any $h(\mathbf{x}) = \mathbf{V}\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, it holds that

$$\|h(\mathbf{x}) - \mathbf{V}r(\mathbf{W}\mathbf{x} + \mathbf{b})\| \leq \|\mathbf{V}\| \cdot q\varepsilon. \tag{45}$$

Moreover, for $\mathbf{y} \in \mathbb{R}^q$, it holds that

$$\begin{aligned} r(\mathbf{y}) &= \begin{pmatrix} r(y_1) \\ r(y_2) \\ \vdots \\ r(y_q) \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^s m_i \sigma(n_i y_1 + p_i) \\ \sum_{i=1}^s m_i \sigma(n_i y_2 + p_i) \\ \vdots \\ \sum_{i=1}^s m_i \sigma(n_i y_q + p_i) \end{pmatrix} \\ &= \mathbf{M}\sigma_R(\mathbf{N}\mathbf{y} + \mathbf{p}), \end{aligned} \tag{46}$$

where

$$\begin{aligned} \mathbf{M} &= \mathbf{I}_q \otimes (m_1, m_2, \cdots, m_s), \\ \mathbf{N} &= \mathbf{I}_q \otimes (n_1, n_2, \cdots, n_s)^T, \\ \mathbf{p} &= \underbrace{(1, \cdots, 1)}_{1 \times q}^T \otimes (p_1, p_2, \cdots, p_s)^T. \end{aligned} \tag{47}$$

Therefore, by Eq. (45) and Eq. (46), we have

$$\|h(\mathbf{x}) - \mathbf{V}\mathbf{M}\sigma_R(\mathbf{N}(\mathbf{W}\mathbf{x}+\mathbf{b})+\mathbf{p})\| \leq \|\mathbf{V}\| \cdot q\varepsilon. \quad (48)$$

Note that $\mathbf{VM} \in \mathbb{R}^{d\times qs}$, from Eq. (48), we have

$$h \in \overline{\tilde{\mathcal{F}}_{\sigma_R}} \subset \overline{\mathrm{CH}}(\tilde{\mathcal{F}}_{\sigma_R}), \quad (49)$$

where

$$\tilde{\mathcal{F}}_{\sigma_R} = \{\mathbf{z} \mapsto \mathbf{V}\sigma_R(\mathbf{W}\mathbf{z}+\mathbf{b}) : \\ \mathbf{V} \in \mathbb{R}^{d\times qs}, \mathbf{W} \in \mathbb{R}^{qs\times d}, \mathbf{b} \in \mathbb{R}^{qs}\}.$$

Since $qs \geq 2$, Proposition 4 holds for the terminal family $\tilde{\mathcal{F}}_{\sigma_R}$. Thus $\mathcal{A}_{\tilde{\mathcal{F}}_{\sigma_R}}$ is L^p-universal in $C([0,1]^d, \mathbb{R}^m)$. Finally from Eq. (49), we have $\mathcal{F}_\sigma \subset \overline{\mathrm{CH}}(\tilde{\mathcal{F}}_{\sigma_R})$. Thus by Proposition 5, it holds that

$$\overline{\mathcal{A}_{\mathcal{F}_\sigma}} = \overline{\mathcal{A}_{\tilde{\mathcal{F}}_{\sigma_R}}}.$$

Hence the universality holds for $\mathcal{F}_\sigma$.

Now we only need to find such a function r. By Assumption 1, σ is L_σ-Lipschitz continuous on $\mathbb{R}$, thus we have

$$|\sigma(x) - \sigma(y)| \leq L_\sigma|x-y|, \quad \forall x, y \in \mathbb{R}.$$

Moreover, by Assumption 1, $\forall \varepsilon > 0$, there exists a constant $M > \max\{|a|, |b|\}$ and affine functions $v_1(x) = c_1x + d_1$ and $v_2(x) = c_2x + d_2$, such that

$$\|\sigma - v_1\|_{C((-\infty,-M])} \leq \frac{\varepsilon}{2}, \quad \|\sigma - v_2\|_{C([M,\infty))} \leq \frac{\varepsilon}{2}.$$

Let $N = \left\lceil \frac{4ML_\sigma}{\varepsilon} \right\rceil$ be an integer. Define $\delta = \frac{2M}{N}$ and $x_i = -M + i\delta$ for $0 \leq i \leq N$. It can be seen that $\delta \leq \frac{\varepsilon}{2L_\sigma}$. For any (x_1, y_1) and $(x_2, y_2) \in \mathbb{R}^2$, $x_1 < x_2$, we can linearly interpolate between them by

$$\frac{y_2 - y_1}{x_2 - x_1}(\sigma_R(x - x_1) - \sigma_R(x - x_2)) + y_1.$$

Now we define

$$k_1(x) = -c_1\sigma_R(-x-M)+\sigma(-M), \quad k_2(x) = c_2\sigma_R(x-M).$$

we further define

$$t(x) = k_1(x) + k_2(x) \\ + \sum_{i=1}^{N} \frac{\sigma(x_i) - \sigma(x_{i-1})}{x_i - x_{i-1}} \left(\sigma_R(x - x_{i-1}) - \sigma_R(x - x_i)\right).$$

When $x \geq M$, by the fact that $|\sigma(M) - v_2(M)| \leq \varepsilon$ and above definitions we have

$$\begin{aligned} &|\sigma(x) - t(x)| \\ =&\left|\sigma(x) - \sigma(-M) - \sum_{i=1}^{N}(\sigma(x_i) - \sigma(x_{i-1})) - c_2(x - M)\right| \\ =&|\sigma(x) - \sigma(M) - c_2x + c_2M| \\ \leq&|\sigma(x) - c_2x - d_2| + |d_2 + c_2M - \sigma(M)| \\ \leq&\varepsilon. \end{aligned}$$

When $x \leq -M$, we also have

$$\begin{aligned} &|\sigma(x) - t(x)| \\ =&|\sigma(x) - c_1x - c_1M - \sigma(-M)| \\ =&|\sigma(x) - c_1x - \sigma(-M) - c_1M| \\ \leq&|\sigma(x) - c_1x - d_1| + |d_1 - c_1M - \sigma(-M)| \\ \leq&\varepsilon. \end{aligned}$$

When $x \in [x_0, x_N]$, there exists $j \in \{1, \cdots, N\}$, such that $x \in [x_{j-1}, x_j]$. Then from the Lipschitz continuity of σ, it holds

$$\begin{aligned} &|\sigma(x) - t(x)| \\ \leq&|\sigma(x) - \sigma(x_j)| + |\sigma(x_j) - t(x_j)| + |t(x_j) - t(x)| \\ \leq&L_\sigma|x - x_j| + |t(x_j) - t(x_{j-1})| \\ \leq&L_\sigma|x - x_j| + |\sigma(x_j) - \sigma(x_{j-1})| \\ \leq&\varepsilon. \end{aligned}$$

Therefore, we can take $r(x) = t(x)$ satisfying Eq. (44), which completes the proof. □

B.3 Proof of Proposition 2

Proof. We will prove that there exists a vector norm and the induced matrix norm $\|\cdot\|_{op}$, such that $\left\|\prod_{k=s}^{s'} \mathbf{P}_k\right\|_{op}$ is uniformly bounded for all $s' > s$, where

$$\mathbf{P}_k = \begin{pmatrix} \alpha_{k-1}^k & \alpha_{k-2}^k & \cdots & \alpha_1^k & \alpha_0^k \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \in \mathbb{R}^{(k+1)\times k}. \quad (50)$$

From the property of the companion matrix [75], we know that the characteristic polynomial of the matrix

$$\tilde{\mathbf{P}}_k = \begin{pmatrix} \alpha_{k-1}^k & \alpha_{k-2}^k & \cdots & \alpha_1^k & \alpha_0^k \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \in \mathbb{R}^{k\times k}.$$

is exactly the $\rho_k(\lambda)$ in Theorem 1. By 3) in Assumption 2, the eigenvalues of $\tilde{\mathbf{P}}_k$ lies within the open unit disc $1-\varepsilon$ except for an eigenvalue 1. Note that $(1, 1, \cdots, 1)^T$ is the eigenvector of all $\tilde{\mathbf{P}}_k$ by 1) in Assumption 2, a transformation of Jordan canonical form yields

$$\mathbf{T}^{-1}\tilde{\mathbf{P}}_k\mathbf{T} = \mathrm{diag}\left\{1, \begin{pmatrix} \lambda_{k,2} & \delta_{k,2} & \\ & \ddots & \delta_{k,k-1} \\ & & \lambda_{k,k} \end{pmatrix}\right\} := \mathbf{J}_k,$$

where $\lambda_{k,2}, \cdots, \lambda_{k,k}$ are the eigenvalues of modulus less than $1-\varepsilon$, and δ_i is either 0 or 1. For any $l > 0$, define

$$\mathbf{D}_l = \mathrm{diag}\{1, l, \cdots, l^{k-1}\} \in \mathbb{R}^{k\times k}.$$

By simple calculation, we have

$$\mathbf{D}_l^{-1}\mathbf{T}^{-1}\tilde{\mathbf{P}}_k\mathbf{T}\mathbf{D}_l = \mathrm{diag}\left\{1, \begin{pmatrix} \lambda_{k,2} & l\delta_{k,2} & \\ & \ddots & l\delta_{k,k-1} \\ & & \lambda_{k,k} \end{pmatrix}\right\}.$$

Note that $|\lambda_{k,j}| \leq 1-\varepsilon$, $\forall k, j$. By choosing l appropriately, we can assume that

$$|\delta_{k,j}| \leq 1 - |\lambda_{k,j}|, \ \forall\, 2 \leq j \leq k, \ \forall\, k.$$

Hence, from the definition of $\|\cdot\|_\infty$, we have

$$\left\|\mathbf{T}^{-1}\tilde{\mathbf{P}}_k\mathbf{T}\right\|_\infty \leq 1, \quad \forall\, k.$$

Now define a vector norm as $\|\mathbf{x}\|_{op} = \|\mathbf{T}^{-1}\mathbf{x}\|_\infty$. Then by definition, it holds that

$$\begin{aligned}\|\tilde{\mathbf{P}}_k\mathbf{x}\|_{op} &= \|\mathbf{T}^{-1}\tilde{\mathbf{P}}_k\mathbf{x}\|_\infty = \|\mathbf{J}_k\mathbf{T}^{-1}\mathbf{x}\|_\infty \\ &\le \|\mathbf{T}^{-1}\mathbf{x}\|_\infty = \|\mathbf{x}\|_{op}.\end{aligned}$$

Therefore, we have $\|\tilde{\mathbf{P}}_k\|_{op} \le 1$. Finally, since $\|\tilde{\mathbf{P}}_k\|_\infty = \|\mathbf{P}_k\|_\infty$ and the fact that vector norms are equivalent in a finite-dimensional space, we have that

$$\left\|\prod_{k=s}^{s'}\mathbf{P}_k\right\|_{op} \le \prod_{k=s}^{s'}\|\mathbf{P}_k\|_{op} \le 1.$$

We finish the proof. □

Remark 2. *When the algorithm $\mathscr{A}$ is a fixed-step algorithm, 3) in Assumption 2 can be relaxed to that $\rho_k(\lambda)$ satisfies the root condition in Definition 5. To see this, from the proof above, we have $\tilde{\mathbf{P}}_k = \tilde{\mathbf{P}}_j$ for all k, j. Then $\left\|\prod_{k=s}^{s'}\tilde{\mathbf{P}}_k\right\|_{op}$ is bounded if all the eigenvalues of $\tilde{\mathbf{P}}_k$ lies within the unit circle and the roots on the unit circle are simple.*

B.4 Proof of Theorem 3

Proof. We want to prove the convergence of the difference method

$$-\sum_{i=0}^{k+1}\alpha_i^{k+1}\mathbf{x}_i = \sum_{i=0}^{k+1}\Delta t\beta_i^{k+1}\mathbf{u}_i^{k+1}, \quad \alpha_{k+1}^{k+1} = -1, \tag{51}$$

when the assumptions of Theorem 1 hold. Define the error as $\mathbf{e}_n = \mathbf{x}(t_n) - \mathbf{x}_n$, and the local truncation error (See Definition 7) for each $\mathbf{x}_{k+1}$ as

$$\begin{aligned}\mathbf{r}_{k+1} &= -\sum_{i=0}^{k+1}\alpha_i^{k+1}\mathbf{x}(t_i) - \sum_{i=0}^{k+1}\beta_i^{k+1}\Delta t\mathbf{u}(t_i,\mathbf{x}(t_i)) \\ &= -\sum_{i=0}^{k+1}[\alpha_i^{k+1}\mathbf{x}(t+i\Delta t) + \Delta t\beta_i^{k+1}\dot{\mathbf{x}}(t+i\Delta t)].\end{aligned}$$

Then from Eq. (51) we have

$$\begin{aligned}-\sum_{i=0}^{k+1}\alpha_i^{k+1}\mathbf{e}_i &= \sum_{i=0}^{k+1}\alpha_i^{k+1}(\mathbf{x}(t_i) - \mathbf{x}_i) \\ &= \sum_{i=0}^{k+1}\beta_i^{k+1}\Delta t(\mathbf{u}(t_i,\mathbf{x}(t_i)) - \mathbf{u}(t_i,\mathbf{x}_i)) + \mathbf{r}_{k+1}.\end{aligned} \tag{52}$$

Further define

$$\mathbf{b}_{k+1} = \sum_{i=0}^{k+1}\beta_i^{k+1}\Delta t(\mathbf{u}(t_i,\mathbf{x}(t_i)) - \mathbf{u}(t_i,\mathbf{x}_i)).$$

Then we can write Eq. (52) as

$$\mathbf{E}_{k+1} = \mathbf{P}_{k+1}\mathbf{E}_k + \mathbf{B}_{k+1},$$

where

$$\begin{aligned}\mathbf{E}_k &= (\mathbf{e}_k, \mathbf{e}_{k-1}, \cdots, \mathbf{e}_0)^T \in \mathbb{R}^{(k+1)\times d}, \\ \mathbf{B}_k &= (\mathbf{b}_k + \mathbf{r}_k, \mathbf{0}, \cdots, \mathbf{0})^T \in \mathbb{R}^{(k+1)\times d},\end{aligned} \tag{53}$$

and $\mathbf{P}_k$ is defined as in Eq. (50). By recursion, we have

$$\mathbf{E}_{k+1} = \prod_{i=1}^{k+1}\mathbf{P}_i\mathbf{E}_0 + \sum_{i=1}^{k+1}\prod_{j=i+1}^{k+1}\mathbf{P}_j\mathbf{B}_i. \tag{54}$$

From Proposition 2, $\|\prod_{i=s}^{s'}\mathbf{P}_i\|_{op}$ is uniformly bounded. By the equivalence of $\|\cdot\|_{op}$ and $\|\cdot\|_\infty$, there exists a constant $c_1 \ge 0$, such that

$$\left\|\prod_{i=s}^{s'}\mathbf{P}_i\right\|_\infty \le c_1, \quad \forall s < s'.$$

Then from Eq. (54), we have

$$\|\mathbf{E}_{k+1}\|_\infty \le c_1\|\mathbf{E}_0\|_\infty + \sum_{i=1}^{k+1}c_1\|\mathbf{B}_i\|_\infty. \tag{55}$$

Now we want to bound $\|\mathbf{B}_i\|_\infty$. For $k \ge k_0$, define the generating polynomial of the k-th step by

$$\rho_k(x) = -\sum_{i=k_0}^{k}\alpha_i^k x^{i-k_0}, \quad \theta_k(x) = \sum_{i=k_0}^{k}\beta_i^k x^{i-k_0}$$

for each $\mathbf{z}_{k+1}$. Since σ satisfies Assumption 1, i.e., it is infinitely differentiable except for at most finitely many points, by Taylor expansion, it holds that

$$\begin{aligned}\mathbf{r}_k =& \sum_{i=0}^{k}\left[-\alpha_i^k\mathbf{x}(t_{k_0}+i\Delta t) - \Delta t\beta_i^k\dot{\mathbf{x}}(t_{k_0}+i\Delta t)\right] \\ =& \sum_{i=0}^{k}\left[-\alpha_i^k(\mathbf{x}(t_{k_0}) + i\Delta t\dot{\mathbf{x}}(t_{k_0}) + o(\|\Delta t\|))\right] \\ &+ \sum_{i=0}^{k}\left[-\Delta t\beta_i^k(\dot{\mathbf{x}}(t_{k_0}) + i\Delta t\ddot{\mathbf{x}}(t_{k_0}) + o(\|\Delta t\|)\right] \\ =& \rho_k(1)\mathbf{x}(t_{k_0}) + (\dot{\rho}_k(1) - \theta_k(1))\,\dot{\mathbf{x}}(t_{k_0})\Delta t + o(\|\Delta t\|).\end{aligned}$$

Therefore, if

$$\rho_k(1) = 0, \quad \dot{\rho}_k(1) = \theta_k(1),$$

then there exists a constant $c_2 > 0$, such that

$$\|\mathbf{r}_i\|_1 \le c_2\Delta t^{p'+1}, \text{ where } p' \ge 1, \ \forall\, i. \tag{56}$$

By 1) in Assumption 2, we already have $\rho_k(1) = 0$. From 2) in Assumption 2, it holds that $\sum_{i=k_0}^{k}|\beta_i^k| \ne 0$. Thus there exists $j \ge k_0$, such that $\beta_j^k \ne 0$. We can modify the value of $\mathbf{V}_j^k$ and β_j^k to meet $\dot{\rho}_k(1) = \theta_k(1)$. Hence, we can find such constant c_2 that Eq. (56) holds.

By the assumption that $\mathbf{u}(t,\mathbf{x})$ is L_u-Lipschitz with respect to $\mathbf{x}$, it holds

$$\|\mathbf{b}_{k+1}\|_1 \le \sum_{i=0}^{k+1}\beta_i^{k+1}\Delta tL_u\|\mathbf{e}_i\|_1.$$

Note that from 2) and 3) in Assumption 2, it can be seen that $|\dot{\rho}_k(1)|$ is uniformly bounded. Thus $|\beta_i^k|$ can be uniformly bounded with the above modification, and we denote the bound by β. For matrix $\mathbf{B}_m$, it holds that

$$\|\mathbf{B}_m\|_\infty \le \beta L_u\Delta t\sum_{i=0}^{m}\|\mathbf{e}_i\|_1 + \|\mathbf{r}_m\|_1,$$

where we use the fact that $\|\mathbf{B}_m\|_\infty = \|\mathbf{b}_m + \mathbf{r}_m\|_1$. Combining it with Eq. (55), we get

$$\|\mathbf{E}_{k+1}\|_\infty \le c_1\|\mathbf{E}_0\|_\infty + c_1\beta L_u\Delta t\sum_{i=1}^{k+1}\sum_{j=0}^{i}\|\mathbf{e}_j\|_1 + c_1\sum_{i=1}^{k+1}\|\mathbf{r}_i\|_1. \tag{57}$$

Denote $\|\mathbf{E}_0\|_\infty$ by ϵ_0. By Eq. (57) and Eq. (56), we have

$$\begin{aligned}
&\|\mathbf{e}_{k+1}\|_1 \le \|\mathbf{E}_{k+1}\|_\infty \\
\le& c_1\epsilon_0 + c_1c_2(k+1)\Delta t^{p'+1} \\
&+ c_1\beta L_u\Delta t(k+1)\sum_{i=0}^{k}\|\mathbf{e}_i\|_1 + c_1\beta L_u\Delta t\|\mathbf{e}_{k+1}\|_1 \\
\le& c_1\epsilon_0 + c_1c_2\Delta t^{p'}\tau + c_1\beta L_u\tau\sum_{i=0}^{k}\|\mathbf{e}_i\|_1 + c_1\beta L_u\Delta t\|\mathbf{e}_{k+1}\|_1,
\end{aligned} \tag{58}$$

where we use $k+1 \le \frac{\tau}{\Delta t}$ in the third equation. When $c_1\beta L_u\Delta t < \frac{1}{2}$, denote $c_0 = \frac{1}{1-c_1\beta L_u\Delta t}$, then it holds

$$\|\mathbf{e}_{k+1}\|_1 \le c_1\epsilon_0 + c_0c_1c_2\Delta t^{p'}\tau + c_0c_1\beta L_u\tau\sum_{i=0}^{k}\|\mathbf{e}_i\|_1.$$

By the Gronwall-Bellman inequality in Lemma 1, we have

$$\|\mathbf{e}_k\|_1 \le \left(c_1\epsilon_0 + c_0c_1c_2\Delta t^{p'}\tau\right)\cdot\frac{e^{c_0c_1\beta L_u\tau}-1}{c_0c_1\beta L_u\tau} \tag{59}$$

for all k. Thus when $\Delta t \to 0$ and $\epsilon_0 \to 0$, $\|\mathbf{e}_k\|_1 \to 0$, which gives the convergence of the method. Moreover, concerning the convergence rate, it only depends on L_u and τ. Thus the convergence holds uniformly with respect to $\mathbf{x}_0$.

□

B.5 Proof of Theorem 1

We give a complete version of the proof of Theorem 1, i.e., we do not assume that $m = d$ or neglect the affine transformation in the last layer of Eq. (16).

Proof. For any function $g \in C([0,1]^d, \mathbb{R}^m)$ and $\forall\varepsilon > 0$, from Proposition 1, there exists a linear surjection $l : \mathbb{R}^d \to \mathbb{R}^m$ and a continuous function $\psi : [0,1]^d \to \mathbb{R}^d$, such that

$$\|g - l\circ\psi\|_{C([0,1]^d,\mathbb{R}^m)} \le \varepsilon. \tag{60}$$

Fix the affine transformation of the last layer in OptDNN in Eq. (15) as l, i.e.,

$$l(\mathbf{z}_L) = W^{L+1}\mathbf{z}_L + b^{L+1} = \mathbf{z}_{L+1}. \tag{61}$$

Note that l is a continuous function on a compact set $[0,1]^d$, therefore $\|l\|_{C([0,1]^d,\mathbb{R}^d)}$ is bounded. Denote $\|l\|_{C([0,1]^d,\mathbb{R}^d)}$ by C, $0 \le C < \infty$.

For function ψ, from Theorem 2, there exists a function $\mathbf{u}(t,\mathbf{z}) = \mathbf{V}^t\sigma(\mathbf{W}^t\mathbf{z} + \mathbf{b}^t)$ and $\tau > 0$, such that

$$\|\psi - h\|_{L^p([0,1]^d,\mathbb{R}^d)} \le \frac{\varepsilon}{C}, \tag{62}$$

where $h(\mathbf{x}) = \mathbf{z}(\tau;\mathbf{x})$, and $\mathbf{z}$ is defined by the following NODE:

$$\dot{\mathbf{z}}(t) = \mathbf{u}(t,\mathbf{z}(t)), \quad \mathbf{z}(0) = \mathbf{x}, \quad t \in [0,\tau]. \tag{63}$$

Now discretize the NODE in Eq. (63) with the mixed LMM of Eq. (22), where parameters α_i^k and β_i^k are determined by the algorithm $\mathscr{A}$. Then from Theorem 3, there exists a $\Delta t > 0$ for all $\mathbf{x}$, such that

$$\|h(\mathbf{z_0}) - \zeta(\mathbf{z}_0)\|_\infty \le \frac{\varepsilon}{C}, \tag{64}$$

where $\zeta(\mathbf{z}_0) = \mathbf{z}_L$ is the function of OptDNN and $\mathbf{z}_L$ is defined by

$$\begin{aligned}
\mathbf{z}_{k+1} =& \sum_{i=0}^{k}\alpha_i^{k+1}\mathbf{z}_i + \sum_{i=0}^{k+1}\beta_i^{k+1}\left(\Delta t\mathbf{V}_i^{k+1}\right)\sigma(\mathbf{W}_i^{k+1}\mathbf{z}_i + \mathbf{b}_i^{k+1}), \\
&k = 0,1,\cdots,L-1, \quad L = \lceil\frac{\tau}{\Delta t}\rceil.
\end{aligned}$$

From Eq. (62) and Eq. (64), we have

$$\|\psi - \zeta\|_{L^p([0,1]^d,\mathbb{R}^d)} \le \frac{2\varepsilon}{C}. \tag{65}$$

Combining Eq. (60), Eq. (61) and Eq. (65), we have

$$\begin{aligned}
&\|g - l\circ\zeta\|^p_{L^p([0,1]^d,\mathbb{R}^m)} \\
&\le \|g - l\circ\psi\|^p_{L^p([0,1]^d,\mathbb{R}^m)} + \|l\circ\psi - l\circ\zeta\|^p_{L^p([0,1]^d,\mathbb{R}^m)} \\
&\le \varepsilon^p + \|l\|_{C([0,1]^d,\mathbb{R}^d)}\cdot\|\psi - \zeta\|^p_{L^p([0,1]^d,\mathbb{R}^d)} \\
&\le 2\varepsilon^p.
\end{aligned}$$

This yields the result. □

B.6 Proof of Proposition 3

We first introduce Lemma 4.7 in [20]. Recall the definition of $\phi_\tau^{\mathbf{u}}$ and $\mathcal{A}_{\mathcal{F}}(\tau)$ in Subsection B.2.

Lemma 2 (Lemma 4.7, [20]). *Let $d = 1$ and $\phi : \mathbb{R} \to \mathbb{R}$ be an increasing continuous increasing differentiable function. Suppose ϕ is piecewise smooth and $T_0 := \|\ln\phi'\|_{\mathrm{TV}([0,1])} \le \infty$. Then $\phi \in$ If $\ln\phi'$ is a piece-wise constant function with N pieces, then ϕ can be written as*

$$\phi = \phi_{t_1}^{g_1}\circ\cdots\circ\phi_{t_{N-1}}^{g_{N-1}} + c,$$

where $g_i \in \mathcal{F}_{\sigma_R}$ and c is a constant. Moreover, we have $\phi \in \mathcal{A}_{\mathcal{F}_{\sigma_R}}(T)$ for $T \ge \|\ln\phi'\|_{\mathrm{TV}}$ when the control $v\sigma_R(wx+b)$ is restricted as $|v|, |w| < 1$. Here $\|\cdot\|_{\mathrm{TV}}$ refers to he total variation norm.

Now we begin our proof. We first consider the rate of approximating g using NODEs, then we consider the effect of discretization.

Proof. **Approximating using NODEs:** Our idea is to approximate g using simple functions in the form of $\sum_i c_i\mathbf{1}_{\Omega^i}$ where Ω^i are some grids. First, we show that NODEs can approximate indicator functions, which enables us to contract a grid into a point. Then we construct a 'projection' map by NODEs to project all the points in a one-dimensional space $\{\mathbf{x}\in\mathbb{R}^d : x_1 = \cdots = x_{d-1} = 0\}$ and separate them on the d-th entry. Since $d > m$, we can utilize the d-th entry to realize piecewise linear interpolation in the first m dimension, and 'truncate' the rest $d-m$ entries by using the linear transformation in the last layer.

$\forall\varepsilon > 0$, let $N > 0$ be an integer to be determined later. We equally partition $[0,1]^d$ into N^d small grids. Denote $s = N^d$ and the grids by $\{\Omega^i\}_{i=1}^s$. Each grid is in the form of $\left[\frac{j_1}{N},\frac{j_1+1}{N}\right]\times\cdots\times\left[\frac{j_d}{N},\frac{j_d+1}{N}\right]$. Now we want to construct NODEs to contract each grid into a point. Define $\varphi : \mathbb{R}\to\mathbb{R}$, $\varphi(x) = \sum_i\frac{1}{N}\mathbf{1}_{x\in[\frac{i}{N},\frac{i+1}{N})}$ and $\tilde{\varphi}(\mathbf{x}) = (\varphi(x_1),\cdots,\varphi(x_d))^T$.

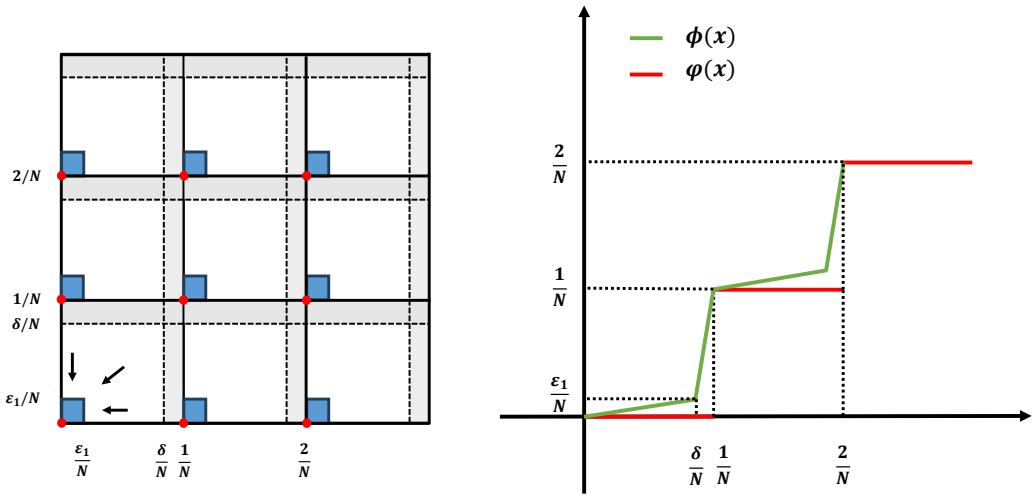


Fig. 7. On the right is the graph of $\varphi(x)$ and $\phi(x)$. On the left is the illustration of $\tilde{\varphi}(\mathbf{x})$ and $\tilde{\phi}(\mathbf{x})$ on $[0,1]^d$. Specifically, $\tilde{\varphi}(\mathbf{x})$ contracts each grid into red point on the left bottom of the grid, while $\tilde{\phi}(\mathbf{x})$ contracts the white part of each grid (i.e. the original grid except for the shaded area) into the small blue neighborhood of the red point.

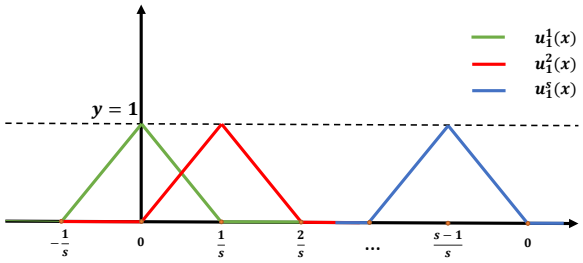


Fig. 8. Illustration of the graph of $u_1^l(x)$.

Let ε_1, δ be some constants to be defined later which satisfy $0 < \varepsilon_1 < \delta < 1$. We further define $\phi : \mathbb{R} \to \mathbb{R}$ as

$$\phi(x) = \begin{cases} \dfrac{\varepsilon_1}{\delta}x + \dfrac{i}{N} - \dfrac{i\varepsilon_1}{N\delta}, & x \in \left[\dfrac{i}{N}, \dfrac{i+\delta}{N}\right], \\ \dfrac{1-\varepsilon_1}{1-\delta}x + t_i, & x \in \left[\dfrac{i+\delta}{N}, \dfrac{i+1}{N}\right], \end{cases}$$

for each $x \in [\frac{i}{N}, \frac{i+1}{N}]$ and $t_i = \frac{i+1}{N} - \frac{(i+1)(1-\varepsilon_1)}{(1-\delta)N}$. The graphs of φ and ϕ and are shown in Figure 7. Since ϕ is an increasing piecewise linear function, by Lemma 2, it holds that $\phi \in \mathcal{A}_{\mathcal{F}_{\sigma_R}}(T)$ for $T \geq \|\ln \phi'\|_{\mathrm{TV}}$. By definition, we have

$$\|\ln \phi'\|_{\mathrm{TV}} = \ln\left[2N\left(\frac{1-\varepsilon_1}{1-\delta} - \frac{\varepsilon_1}{\delta}\right)\right] \tag{66}$$

and we can just let $T = \|\ln \phi'\|_{\mathrm{TV}}$. Moreover, it holds that

$$|\phi(x) - \varphi(x)| \leq \frac{\varepsilon_1}{N}, \quad x \in \left[\frac{i}{N}, \frac{i+\delta}{N}\right]. \tag{67}$$

By using concatenation, we can form a d-dimensional map in $\mathcal{A}_{\mathcal{F}_{\sigma_R}}(T)$ defined as $\tilde{\phi}(\mathbf{x}) = (\phi(x_1), \phi(x_2), \cdots, \phi(x_d))^T$. Note that $\tilde{\phi}$ approximately maps every $\mathbf{x} \in \left[\frac{j_1}{N}, \frac{j_1+1}{N}\right) \times \cdots \times \left[\frac{j_d}{N}, \frac{j_d+1}{N}\right)$ to a point $\left(\frac{j_1}{N}, \cdots, \frac{j_d}{N}\right)^T$.

Next, we construct a projection map to separate the contracted points $\{\mathbf{x}^i\}_i$, i.e., $\tilde{\varphi}(\mathbf{x}) = \mathbf{x}^i$ for $\mathbf{x} \in \Omega^i$. Inspired by binary representation, we consider a map π induced by the following dynamics

$$\dot{z}_d(t) = \sum_{i=1}^{d-1} N^{-i} z_i, \quad \dot{z}_k = 0, \quad 1 \leq k \leq d-1, \quad t \in [0,1], \tag{68}$$

with $\mathbf{z}(0) = \mathbf{x}$. We have

$$\pi(\mathbf{x}) = \mathbf{z}(1; \mathbf{x}), \quad z_d(1) = \sum_{i=1}^{d-1} N^{-i} x_i + x_d.$$

Moreover, dynamic Eq. (68) can be realized by NODEs as $\dot{\mathbf{z}}(t) = \mathbf{V^0}\sigma(\mathbf{W^0}\mathbf{z})$ where

$$\begin{aligned} \mathbf{V}^0(i,j) &= \delta_{d1} + \delta_{d2}, \\ \mathbf{W}^0(i,j) &= (\delta_{1j}N^{-j} - \delta_{2j}N^{-j})\mathbf{1}_{j \leq d-1}. \end{aligned}$$

Here $\mathbf{V}^0(i,j)$ and $\mathbf{W}^0(i,j)$ refers to the element of row i and column j of matrix $\mathbf{V}^0$ and $\mathbf{W}^0$, respectively, and δ is Kronecker delta defined as $\delta_{ij} = \mathbf{1}_{i=j}$. Therefore, we have $\pi \circ \tilde{\phi} \in \mathcal{A}_{\mathcal{F}_{\sigma_R}}(T+1)$. Furthermore, one can verify that for all $\mathbf{x}^i \neq \mathbf{x}^j \in \left\{0, \frac{1}{N}, \frac{2}{N}, \cdots, \frac{N-1}{N}\right\}^d$, it holds that $\pi(\tilde{\varphi}(\mathbf{x}^i)) \neq \pi(\tilde{\varphi}(\mathbf{x}^j))$. In this way, we project all the contracted points to the one-dimensional space $\{\mathbf{x} \in \mathbb{R}^d : x_1 = \cdots = x_{d-1} = 0\}$ and separate them on the d-th entry.

Now we aim at constructing an NODE whose realization $\mathbf{h}$ satisfies $h_k(\pi(\tilde{\varphi}(\mathbf{x}^i))) = g_k(\mathbf{x}^i)$ for $1 \leq k \leq m$. For simplicity, denote $\pi(\tilde{\varphi}(\mathbf{x}^i))$ by $\tilde{\mathbf{x}}^i$ and N^{d-1} by s', respectively. Note that

$$(\pi \circ \tilde{\varphi})_d(\mathbf{x}^i) \in \left\{0, \frac{1}{s'}, \frac{2}{s'}, \cdots, \frac{N^d - 1}{s'}\right\}.$$

Without loss of generality, we assume that $\tilde{\mathbf{x}}_d^1 < \cdots < \tilde{\mathbf{x}}_d^s$ and $\tilde{\mathbf{x}}_d^i = \frac{i-1}{s'}$. For $k = 1$, we claim that there exist a set of functions $\{\mathbf{u}^l(\mathbf{x})\}_{l=1}^s$ such that $\mathbf{u}^l(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$,

$\mathbf{u}^l(\mathbf{x}) = \mathbf{V}^l\sigma(\mathbf{W}^l\mathbf{x} + \mathbf{b}^l)$ and $\mathbf{u}^l(\mathbf{x}) = (u_1^l(x_d), 0, \cdots, 0)^T$, where $u_1^l(x) : \mathbb{R} \to \mathbb{R}$ is defined as follows

$$u_1^l(x) = \begin{cases} s'x - l + 1, & x \in \left[\frac{l-2}{s'}, \frac{l-1}{s'}\right], \\ -s'x + l + 1, & x \in \left[\frac{l-1}{s'}, \frac{l}{s'}\right], \\ 0, & x \in \left(-\infty, \frac{l-2}{s'}\right] \cup \left[\frac{l}{s'}, \infty\right). \end{cases}$$

The graph of $u_1^l(x)$ is shown in Figure 8. From the construction of $u_1^l(x)$, we have

$$u_1^l(\tilde{x}_d^l) = 1, \quad u^l(\tilde{x}_d^i) = 0, \quad \forall i \neq l.$$

For the claim, we just need to define

$$\begin{aligned} \mathbf{V}^l(i,j) &= \delta_{1i}\delta_{1j} - \delta_{1i}\delta_{2j} + \delta_{1i}\delta_{3j}, \\ \mathbf{W}^l(i,j) &= s'\delta_{1i}\delta_{dj} + 2s'\delta_{2i}\delta_{dj} + s'\delta_{3i}\delta_{dj}, \\ \mathbf{b}^l(i) &= (-l+2)\delta_{1i} - (2l-2)\delta_{2i} - l\delta_{3i}. \end{aligned}$$

With simple calculation we have $u^l(\mathbf{x}) = \mathbf{V}^l\sigma(\mathbf{W}^l\mathbf{x} + \mathbf{b}^l)$. Now we can construct $\mathbf{h}$ recursively to realize the 'interpolation'.

1) Consider the dynamics

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{u}^1(\mathbf{z}), \quad \mathbf{z}(0) = \tilde{\mathbf{x}}, \quad t \in [0, \tau_1].$$

By the definition of u^1, it is equivalent to

$$\dot{z_1}(t) = u_1^1(z_d), \quad \dot{z_j}(t) = 0, \quad 2 \le j \le d, \quad t \in [0, \tau_1].$$

The ODE could be solved directly as

$$z_1(t) = u_1^1(z_d) \cdot t + \tilde{x}_1, \quad z_j(t) = \tilde{x}_j, \quad 2 \le j \le d. \tag{69}$$

Set $\tau_1 = g_1(\mathbf{x}^1) - \tilde{x}_1^1$, then we have

$$\begin{aligned} z_1(\tau_1; \tilde{x}_1^1) &= g_1(\tilde{\mathbf{x}}^1), \\ z_j(\tau_1; \tilde{x}_j) &= \tilde{x}_j, \quad 2 \le j \le d, \quad \forall \tilde{\mathbf{x}}. \end{aligned}$$

2) Suppose we have constructed a flow map $\mathbf{h}(\cdot)$ such that $h_1(\tilde{\mathbf{x}}^i) = g_1(\mathbf{x}^i), 1 \le i \le l-1$ for $1 < l < s$. Consider the dynamic

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = u^l(\mathbf{z}), \quad \mathbf{z}(0) = \mathbf{h}(\tilde{\mathbf{x}}), \quad t \in [0, \tau_l],$$

namely,

$$\dot{z_1}(t) = u_1^l(z_2), \quad \dot{z_k}(t) = 0, \quad 2 \le k \le d, \quad t \in [0, \tau_l].$$

Take $\tau_l = g_1(\mathbf{x}^l) - \mathbf{h}_1(\tilde{\mathbf{x}}^l)$, since $u_1^l(\tilde{x}_d^l) = 1$, we have

$$z_1(\tau_l; \mathbf{h}(\tilde{\mathbf{x}}^l)) = g_1(\tilde{\mathbf{x}}^l).$$

Note that $u_1^l(\tilde{x}_1^i) = 0$ for all $i \neq l$, thus we have

$$z_1(\tau_l; \mathbf{h}(\tilde{\mathbf{x}}^i)) = g_1(\mathbf{x}^i), \quad 1 \le i \le k.$$

Moreover, the map $\mathbf{z} \circ \mathbf{h}$ will not change the value of other entries (except for the first entry) on all $\tilde{\mathbf{x}}^i$.

By recursion, we could construct a flow map $\mathbf{h}$ as the realization of an NODE, such that

$$\begin{aligned} h_1(\tilde{\mathbf{x}}^i) &= g_1(\mathbf{x}^i), \quad 1 \le i \le s, \\ h_k(\tilde{\mathbf{x}}) &= \tilde{x}_k, \quad 2 \le k \le d, \quad \forall \tilde{\mathbf{x}}. \end{aligned}$$

In the constructing process, we only use the fact that all $\{\tilde{x}_d^i\}_i$ are distinct to construct $\{\mathbf{u}^l(\mathbf{x})\}_l$, thus we can use the same technique to 'interpolate' on coordinate $2, 3, \cdots, m$. With a small abuse of notations, we denote the composition of all the flow maps by $\mathbf{h}$. Hence we construct a flow map $\mathbf{h}$ as the realization of an NODE such that $h_k(\tilde{\mathbf{x}}^i) = g_k(\mathbf{x}^i), 1 \le i \le N^d$ for all $1 \le k \le m$. Finally, we set the linear transformation $\tilde{\mathbf{l}}$ in the last as a constrains onto the first m dimension of $\mathbf{h}$, i.e., $\tilde{l} : \mathbb{R}^d \to \mathbb{R}^m$, $l(\mathbf{x}) = (x_1, \cdots, x_m)^T$.

Now we estimate $\|\tilde{l} \circ \mathbf{h} \circ \pi \circ \tilde{\phi} - \mathbf{g}\|_\infty$. Without loss of generality, we just estimate $|h_1 \circ \pi \circ \tilde{\phi}(\mathbf{x}) - g_1(\mathbf{x})|$. Denote

$$K^\delta = \left\{x \in [0,1]^d : x_k \in \left[\frac{i}{N}, \frac{i+\delta}{N}\right], \forall k\right\},$$

and $[0,1]^d$ by K for simplicity. Then

$$\begin{aligned} &\|h_1 \circ \pi \circ \tilde{\phi} - g_1\|_{L^p(K)} \\ \le &\|h_1 \circ \pi \circ \tilde{\phi} - h_1 \circ \pi \circ \tilde{\varphi}\|_{L^p(K)} + \|h_1 \circ \pi \circ \tilde{\varphi} - g_1\|_{L^p(K)} \\ \le &\|h_1 \circ \pi \circ \tilde{\varphi} - g_1\|_{L^p(K)} + \|h_1 \circ \pi \circ \tilde{\phi} - h_1 \circ \pi \circ \tilde{\varphi}\|_{L^p(K^\delta)} \\ &+ \|h_1 \circ \pi \circ \tilde{\phi} - h_1 \circ \pi \circ \tilde{\varphi}\|_{L^p(K \setminus K^\delta)} \\ := &J_1 + J_2 + J_3. \end{aligned}$$

For J_1, $\forall \mathbf{x} \in \left[\frac{j_1}{N}, \frac{j_1+1}{N}\right) \times \cdots \times \left[\frac{j_d}{N}, \frac{j_d+1}{N}\right)$ and $\mathbf{x}^j = \left(\frac{j_1}{N}, \cdots, \frac{j_d}{N}\right)^T$, from the definition of $\tilde{\varphi}$, we have $\tilde{\varphi}(\mathbf{x}) = \mathbf{x}^j$. Thus

$$|h_1 \circ \pi \circ \tilde{\varphi}(\mathbf{x}) - g_1(\mathbf{x})| \le |g_1(\mathbf{x}^j) - g_1(\mathbf{x})| \le \frac{L_g}{N}.$$

For J_2, from Eq. (69), we know that for $\tilde{\mathbf{x}} \in [0,1]^d$ with $\mathbf{x}_d \in [\frac{l-1}{s'}, \frac{l}{s'}]$,

$$\begin{aligned} h_1(\tilde{\mathbf{x}}) = &\tilde{x}_1 + (-s'\tilde{x}_d + l + 1) \cdot (g_1(\mathbf{x}^l) - \tilde{x}_1^l) \\ &+ (s'\tilde{x}_d - l) \cdot (g_1(\mathbf{x}^{l+1}) - \tilde{x}_1^{l+1}), \quad \tilde{x}_1 = 0. \end{aligned}$$

Hence $h_1(\tilde{\mathbf{x}})$ is an s'-Lipschitz piecewise linear function with respect to $\tilde{x}_d$, and π is 1-Lipschitz with respect to x_d by definition. By Eq. (67), we have $\|\tilde{\phi} - \tilde{\varphi}\|_{C(K^\delta)} = \frac{\varepsilon_1}{N}$. Then we have

$$J_2 \le s\|\tilde{\phi} - \tilde{\psi}\|_{C(K^\delta)} = \frac{s'\varepsilon_1}{N}.$$

For J_3, since the image of π is in $[0,1]^d$ and $h_1(\tilde{\mathbf{x}})$ is piecewise linear in each interval $[\frac{l-1}{s'}, \frac{l}{s'}]$ with respect to $\tilde{x}_d$, we have $\|h_1\|_{C(K)} \le \|g_1\|_{C(K)}$ and thus

$$J_3 \le \|g\|_{C(K)} \cdot |K \setminus K^\delta| \le \|g\|_{C(K)}(1 - \delta^d).$$

In all, we choose $N = \frac{1}{3L_g\varepsilon}$, $\varepsilon_1 = \frac{\varepsilon}{3N^{d-2}}$ and $\delta = \left(\frac{1-\varepsilon}{3\|g\|_{C(K)}}\right)^{\frac{1}{d}}$, then we have

$$\|h_1 \circ \pi \circ \tilde{\phi} - g_1\|_{L^p(K)} \le J_1 + J_2 + J_3 \le \varepsilon.$$

Discretization. Now we consider the discretization. Assume that d is large enough and the step size $\Delta t \ll \frac{1}{N}$. For the NODE corresponding to $\tilde{\phi}$, from Eq. (59) and Eq. (66), the discretization error err_1 is bounded as

$$err_1 \le \frac{c_2}{\beta} \cdot \Delta t^p \left(e^{\|\ln \phi'\|_{\mathrm{TV}}}\right)^{c_0 c_1 \beta L_u},$$

where $c_0 < 1$,$L_u \leq 1$ and we use the fact that $\varepsilon_0 = 0$ in Eq. (59). By Eq. (66), it holds

$$e^{\|\ln \phi'\|_{\mathrm{TV}}} \leq e^{\ln 2N(1-\delta)^{-1}} \leq 2N(1-\delta)^{-1} = \mathcal{O}(d\varepsilon^{-2}),$$

which does not suffer from the curse of dimensionality. Thus err_1 is bounded by

$$err_1 \leq \frac{c_2}{\beta}\cdot\Delta t^p \left(2N(1-\delta)^{-1}\right)^{c_0 c_1 \beta L_u} \leq \mathcal{O}(\Delta t^p d^{c_1\beta}\varepsilon^{-2c_1\beta}).$$

For discretizing the NODE corresponding to π and $\mathbf{h}$, the error err_1 can be treated as the initial error ε_0. Note that the map $z_d(t)$ in Eq. (68) and $z_1(t)$ in Eq. (69) are actually linear, which corresponds to the case that $\mathbf{B}_i = 0$ in Eq. (52). Thus the error will not accumulate in the discretization process except for the points near the endpoints of each constant control. This is to say, the error only occurs at finitely many points near some τ that $u(t,\mathbf{x}) = u(\mathbf{x})$ for $t < \tau$ and $u(t,\mathbf{x}) = v(\mathbf{x}) \neq u(\mathbf{x})$ for $t > \tau$. By 3) in Assumption 2 and Vieta's Theorem, we have $\alpha := \sup_{k,i} \alpha_i^k < \infty$. Then from Eq. (52), the error can be bounded by $k_0^2\alpha\beta\Delta t\|u-v\|_\infty$, where k_0 is defined in Assumption 2 and $\beta = \sup_{k,i}\beta_i^k$. Specifically, for the term $\|u-v\|_\infty$, we have

$$\|u^1 - \pi\|_\infty \leq 2, \quad \|u^l - u^{l+1}\|_\infty \leq 2,$$

and there are N^d endpoints where the error may accumulate. Therefore, the overall discretization error is $\mathcal{O}(\Delta t^p d^{c_1\beta}\varepsilon^{-2c_1\beta}) + \mathcal{O}(\varepsilon^{-d}\Delta t)$. By choosing $\Delta t = \mathcal{O}(\varepsilon^{-d+1})$, the overall error is $\mathcal{O}(\varepsilon)$. Then the depth of the OptDNN L is

$$L = \frac{T}{\Delta t} + \frac{1}{\Delta t} + \frac{N^m\|g-\mathrm{id}\|_\infty}{\Delta t} = \mathcal{O}(\varepsilon^{-m-d+1}).$$

Furthermore, from the definition of $\tilde{\phi}$, π and $\mathbf{h}$, it can be seen that there are at most d entries of matrice $\mathbf{V}, \mathbf{W}$ or $\mathbf{b}$ that are nonzero in the NODEs. In OptDNNs, from the proof of Theorem 3, we can modify the values of β_i^k and assume that $\beta_i^k \neq 0$ only if $i = k$, which is enough to meet the condition that $\dot{\rho}_k(1) = \theta_k(1)$. In this way, the weight of the OptDNN derived by discretizing the corresponding NODE is bounded by $dL = \mathcal{O}(\varepsilon^{-m-d+1})$. We finish the proof. □

B.7 Discussions of the Proofs

At the end of this section, we give reasons why Assumption 2 is reasonable. Besides, we also discuss the connections between the approximation capabilities of NODEs and OptDNNs.

B.7.1 Discussion of Assumption 2

We explain that some conditions in Assumption 2 can be naturally satisfied by most optimization algorithms as follows.

Condition 1). By letting $\mathbf{x} \to \mathbf{x}^*$ in Eq. (2), we derive that $\sum_{i=0}^k \alpha_i^{k+1} = 1$. Thus this condition just requires that if the initial value is $\mathbf{x}^*$, then the generated sequence sticks to $\mathbf{x}^*$.

Condition 2). It can be seen in Subsection 3.2 that many algorithms can be equivalently written in the form where this assumption holds.

Condition 3). From the proof of Proposition 2 and Theorem 3, this condition is used to prove the uniform boundedness of $\left\|\prod_{i=s}^{s'}\mathbf{P}_i\right\|$. Now turn back to Eq. (2), since we have $\sum_{i=0}^k \alpha_i^{k+1} = 1$, it holds that

$$\mathbf{x}_{k+1} - \mathbf{x}^* = \sum_{i=0}^k \alpha_i^{k+1}(\mathbf{x}_i - \mathbf{x}^*) + \sum_{i=0}^{k+1}\beta_i^{k+1}\nabla f(\mathbf{x}_i).$$

Define $\tilde{\mathbf{e}}_n = \mathbf{x}_n - \mathbf{x}^*$, $\tilde{\mathbf{b}}_n = \sum_{i=0}^n \beta_i^n \nabla f(\mathbf{x}_i)$, then we have

$$\tilde{\mathbf{e}}_{k+1} = \sum_{i=0}^k \alpha_i^{k+1}\tilde{\mathbf{e}}_i + \tilde{\mathbf{b}}_{k+1}.$$

This is equivalent to

$$\tilde{\mathbf{E}}_{k+1} = \mathbf{P}_{k+1}\tilde{\mathbf{E}}_k + \tilde{\mathbf{B}}_{k+1},$$

where

$$\tilde{\mathbf{E}}_k = (\tilde{\mathbf{e}}_k, \tilde{\mathbf{e}}_{k-1}, \cdots, \tilde{\mathbf{e}}_0)^T \in \mathbb{R}^{(k+1)\times d},$$
$$\tilde{\mathbf{B}}_k = (\tilde{\mathbf{b}}_k, \mathbf{0}, \cdots, \mathbf{0})^T \in \mathbb{R}^{(k+1)\times d},$$

and $\mathbf{P}_k$ is defined as in Eq. (50). By recursion, we have

$$\tilde{\mathbf{E}}_{s'} = \prod_{i=s}^{s'}\mathbf{P}_i\tilde{\mathbf{E}}_s + \sum_{i=s}^{s'}\prod_{j=s+1}^{s'}\mathbf{P}_j\tilde{\mathbf{B}}_i. \tag{70}$$

By letting f be strongly convex and $\nabla f \to \mathbf{0}$, it can be seen that the boundedness of $\left\|\prod_{i=s}^{s'}\mathbf{P}_i\right\|$ is nearly a necessary condition for the linear convergence of the algorithm (if we allow arbitrary $\tilde{\mathbf{E}}_s$). From the perspective of optimization algorithm design, one also usually requires that $\rho(\mathbf{P}_i) \leq 1$ to achieve a faster convergence rate and stronger robustness [46], where $\rho(\cdot)$ refers to the spectral radius. This corresponds to Condition 3) in our assumption.

B.7.2 The Expressive Power of NODEs and OptDNNs

In our proofs in Section 4, we study the representation capability of OptDNNs following the paradigm of first analyzing the representation capability of NODEs and then analyzing the effect of discretization using LMMs. In other words, we show that OptDNNs are at least as expressive as NODEs. A natural question is if the approximation power of OptDNNs is 'governed' by that of NODEs[10]. Here we give a negative result to this question for ResNet which is a special example of OptDNN. In fact, using the proof techniques in [7], it can be shown that ResNet is a universal approximator in the sense of C-norm, whereas its corresponding NODE is not universal measured in C-norm [48]. Another simpler example is that when the input dimension $d = 1$, NODEs can only generate increasing maps [20], whereas ResNet can approximate decreasing functions up to any precision.

For other OptDNN networks, there are limited results about the approximation capabilities of other networks with general skip connections, especially in a width-bounded setting. In [38], the authors show that the Momentum ResNet (which can be regraded as variant of OptDNN-HB) can learn any linear mapping up to a multiplicative factor,

10. Here,the approximation power just refer to the ability of NODEs or OptDNNs to approximate specific functions up to any precision, regardless of the approximation rate. In fact, the approximation rate of NODEs is usually measured in time horizon, while the approximation rate of layer-based network is usually measured in depth, width or weight. Strictly, their approximation rates are not comparable.

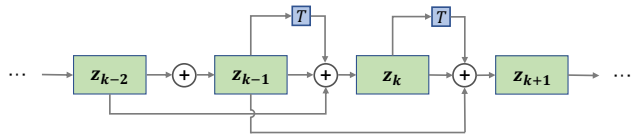


Fig. 9. The architecture of OptDNN-HB.

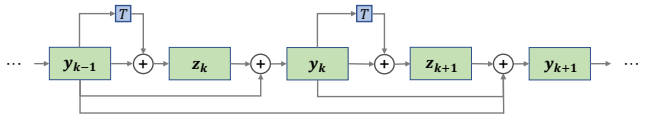


Fig. 10. The architecture of OptDNN-AGD1.

while ResNet cannot. This is likely indicating that OptDNN-HB is also more expressive than NODEs. When the skip connections become more complex, it is more difficult to analyze the approximation capabilities of the network using conventional approaches. Though our analysis methods may not exhaustively explore the full expressive power of OptDNNs, they have allowed us to prove the universality and derive the approximation rate of a rich class of networks with general skip connections.

APPENDIX C
EXPERIMENT DETAILS

C.1 More Examples of OptDNNs

In this subsection, we will show how to design OptDNN from various first-order algorithms. We consider 8 examples of explicit OptDNN and 3 examples of implicit OptDNN. The details of the optimization algorithms can be found in [78] or [79]. In the following, if not specified, we will consider the optimization problem

$$\min_{\mathbf{x}} f(\mathbf{x})$$

and design OptDNNs from it.

- **OptDNN-HB (The Heavy Ball Algorithm):**
  The heavy ball algorithm consists of the following iterations:

  $$\mathbf{x}_{k+1} = \mathbf{x}_k - \beta\nabla f(\mathbf{x}_k) + \alpha(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

  The inspired network (OptDNN-HB) can be represented as:

  $$\mathbf{z}_{k+1} = \mathbf{z}_k + \beta T^k(\mathbf{z}_k) + \alpha(\mathbf{z}_k - \mathbf{z}_{k-1}).$$

  The architecture of OptDNN-HB is shown in Figure 9. With the common parameter choice that $\alpha \in (0, 1)$, this algorithm satisfies Assumption 2. Thus OptDNN-HB is universal by Theorem 1.
- **OptDNN-AGD1 (Nesterov's Accelerated Gradient Descent Algorithm):**
  Nesterov's accelerated gradient descent algorithm consists of iterations:

  $$\begin{aligned}\mathbf{y}_k &= \mathbf{x}_k + \frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}(\mathbf{x}_k - \mathbf{x}_{k-1}),\\ \mathbf{x}_{k+1} &= \mathbf{y}_k - \nabla f(\mathbf{y}_k),\end{aligned}$$

  where θ_k satisfies

  $$\theta_0 = 1, \quad \frac{1-\theta_k}{\theta_k^2} = \frac{1}{\theta_{k-1}^2}. \tag{71}$$

  The inspired network (OptDNN-AGD1) can be represented as:

  $$\begin{aligned}\mathbf{y}_k &= \mathbf{z}_k + \frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}(\mathbf{z}_k - \mathbf{z}_{k-1}),\\ \mathbf{z}_{k+1} &= \mathbf{y}_k + T^k(\mathbf{y}_k).\end{aligned} \tag{72}$$

  The architecture of OptDNN-AGD1 is shown in Figure 10. Note that Nesterov's accelerated gradient descent algorithm is equivalent to

  $$\begin{aligned}&\mathbf{y}_k = \mathbf{y}_{k-1} - \nabla f(\mathbf{y}_{k-1})\\ &+ \frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}(\mathbf{y}_{k-1} - \nabla f(\mathbf{y}_{k-1}) - \mathbf{y}_{k-2} + \nabla f(\mathbf{y}_{k-2})).\end{aligned}$$

  It can be easily verified that 1) and 2) in Assumption 2 holds for this algorithm. For 3), we just need to examine whether the modulus of $\frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}$ is uniformly bounded by $1-\varepsilon$ for some $\varepsilon > 0$. By the definition of θ_k, we have

  $$\theta_{k+1} = \frac{\sqrt{\theta_k^4 + 4\theta_k^2} - \theta_k^2}{2}.$$

  With simple calculations, we have $\theta_k \in [0, 1]$ for all k. Thus it holds $\theta_{k+1} \leq \theta_k$. From this we see that $\frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}$ lies in an open unit circle with radius $1-\varepsilon$. Therefore, Assumption 2 holds for this algorithm, and the OptDNN-AGD1 is universal by Theorem 1.
- **OptDNN-AGD2 (Nesterov's Accelerated Gradient Descent Algorithm):**
  An equivalent form of Nesterov's accelerated gradient descent algorithm is:

  $$\mathbf{y}_{k+1} = \mathbf{y}_k - \sum_{j=0}^{k} h_{k+1,j}\nabla f(\mathbf{y}_j), \tag{73}$$

  where

  $$h_{k+1,j} = \begin{cases}\frac{\theta_{k+1}(1-\theta_k)}{\theta_k}h_{k,j}, \ j = 0, \cdots, k-2,\\ \frac{\theta_{k+1}(1-\theta_k)}{\theta_k}\left(h_{k,k-1} - 1\right), \ j = k-1,\\ 1 + \frac{\theta_{k+1}(1-\theta_k)}{\theta_k}, \ j = k,\end{cases} \tag{74}$$

  and θ is defined as in Eq. (71). The inspired network can be represented as:

  $$\mathbf{z}_{k+1} = \mathbf{z}_k + \sum_{j=0}^{k} h_{k+1,j}T_j^k(\mathbf{x}_j),$$

  where $\{h_{k,j}\}$ are defined as in Eq. (74). The architecture of OptDNN-AGD2 is shown in Figure 3(b).

It can be easily verified that this algorithm satisfies Assumption 2, thus OptDNN-AGD2 is universal by Theorem 1. Although OptDNN-AGD1 and OptDNN-AGD2 are inspired by the same algorithm, it is shown that they perform differently in the experiment (See Table 5 and 6).

- **OptDNN-APG1 (Accelerated Proximal Gradient Method 1):**
We write the primal optimization problem $\min_{\mathbf{x}} f(\mathbf{x})$ equivalently as a composite optimization problem in order to use the accelerated proximal gradient method:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \underbrace{f(\mathbf{x}) - \frac{\|\mathbf{x}\|_2^2}{2}}_{f_1(\mathbf{x})} + \underbrace{\frac{\|\mathbf{x}\|_2^2}{2}}_{f_2(\mathbf{x})}, \tag{75}$$

where $f_1(\mathbf{x}) = f(\mathbf{x}) - \frac{1}{2}\|\mathbf{x}\|_2^2$ and $f_2(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2$. The first sort of the accelerated proximal gradient method consists of iterations:

$$\begin{aligned}
\mathbf{y}_k &= \mathbf{x}_k + \frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}(\mathbf{x}_k - \mathbf{x}_{k-1}),\\
\mathbf{x}_{k+1} &= \operatorname*{argmin}_{\mathbf{x}} \left(f_2(\mathbf{x}) + \frac{1}{2}\|\mathbf{x} - \mathbf{y}_k + \nabla f_1(\mathbf{y}_k)\|_2^2 \right)\\
&= \frac{1}{2}\mathbf{y}_k - \frac{1}{2}\nabla f_1(\mathbf{y}_k).
\end{aligned}$$

The inspired network can be represented as:

$$\mathbf{z}_{k+1} = \beta_k \mathbf{z}_k + \frac{1}{2}\left(\sum_{j=0}^{k} h_{k+1,j} T_j^k(\mathbf{z}_j) - \sum_{j=0}^{k} h_{k+1,j}\mathbf{z}_j \right),$$

where $\beta_k = 1 - \sum_{j=0}^{k} h_{k+1,j}$, and $\{h_{k,j}\}$ are defined as in Eq. (74). The architecture of OptDNN-AGD2 is similar to that of the AGD-NN in [18], which is shown in Figure 3(a). Specifically, the topology of the two nets are the same, whereas the modulating coefficients are different.

- **OptDNN-APG2 (Accelerated Proximal Gradient Method 2):**
Also write the primal problem as Eq. (75), the second sort of accelerated proximal gradient method consists of iterations:

$$\begin{aligned}
\mathbf{y}_k &= \theta_k \mathbf{z}_k + (1-\theta_k)\mathbf{x}_k,\\
\mathbf{z}_{k+1} &= \operatorname*{argmin}_{\mathbf{z}} f_2(\mathbf{z}) + \langle \nabla f_1(\mathbf{y}_k), \mathbf{z}\rangle\\
&\quad + \frac{\theta_k}{2}\left\| \mathbf{z} - \frac{1}{\theta_k}\mathbf{y}_k + \frac{1-\theta_k}{\theta_k}\mathbf{x}_k \right\|_2^2,\\
\mathbf{x}_{k+1} &= (1-\theta_k)x_k + \theta_k \mathbf{z}_{k+1}.
\end{aligned} \tag{76}$$

The inspired network can be represented as:

$$\begin{aligned}
\mathbf{y}_k &= \theta_k \mathbf{z}_k + (1-\theta_k)\mathbf{x}_k,\\
\mathbf{z}_{k+1} &= \frac{2}{1+\theta_k}\mathbf{y}_k + \frac{\theta_k - 1}{1+\theta_k}\mathbf{x}_k + T^k(\mathbf{y}_k),\\
\mathbf{x}_{k+1} &= (1-\theta_k)x_k + \theta_k z_{k+1}.
\end{aligned}$$

The architecture of OptDNN-APG2 is shown in Figure 6(a).

- **OptDNN-APG3 (Accelerated Proximal Gradient Method 3):**
Still writing the primal problem as Eq. (75), another sort of accelerated proximal gradient method consists of iterations:

$$\begin{aligned}
\mathbf{y}_k &= \theta_k \mathbf{z}_k + (1-\theta_k)\mathbf{x}_k,\\
\phi_{k+1}(\mathbf{x}) &= \sum_{i=0}^{k} \frac{f_1(\mathbf{y}_i) + \langle \nabla f(\mathbf{y}_i), \mathbf{x} - \mathbf{y}_i\rangle + f_2(\mathbf{x})}{\theta_i},\\
\mathbf{z}_{k+1} &= \operatorname*{argmin}_{\mathbf{z}} \left(\phi_{k+1}(\mathbf{z}) + \frac{L}{2}\|\mathbf{z} - \mathbf{x}_0\|_2^2 \right),\\
\mathbf{x}_{k+1} &= (1-\theta_k)\mathbf{x}_k + \theta_k \mathbf{z}_{k+1}.
\end{aligned}$$

The inspired network can be represented as

$$\begin{aligned}
\mathbf{y}_k =& \theta_k \mathbf{z}_k + (1-\theta_k)\mathbf{x}_k,\\
\mathbf{z}_{k+1} =& \frac{\mathbf{x}_0}{1 + \sum_{i=0}^{k}\theta_i} - \sum_{i=0}^{k} \frac{\mathbf{y}_i}{\left(1 + \sum_{j=0}^{k}\theta_j\right)\theta_i}\\
&+ \sum_{i=0}^{k} T_i^k(\mathbf{y}_i),\\
\mathbf{x}_{k+1} =& (1-\theta_k)\mathbf{x}_k + \theta_k \mathbf{z}_{k+1}.
\end{aligned}$$

The architecture of OptDNN-APG3 is shown in Figure 6(b).

- **OptDNN-LADMM1 (Linearized ADMM 1):**
We write the primal problem $\min_{\mathbf{x}} f(\mathbf{x})$ and equivalently as the following problem in order to use the linearized ADMM:

$$\min_{\mathbf{x},\mathbf{y}} f_1(\mathbf{x}) + f_2(\mathbf{y}), \quad s.t. \quad \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{b}, \tag{77}$$

where we set $f_1 = f_2 = f$ and $\mathbf{A} = \mathbf{I}, \mathbf{B} = -\mathbf{I}, \mathbf{b} = \mathbf{0}$.
The first sort of linearized ADMM consists of iterations:

$$\begin{aligned}
\mathbf{x}_{k+1} =& \operatorname{Prox}_{(\beta\|\mathbf{A}\|_2^2)^{-1} f_1}\Big(\mathbf{x}_k\\
&- (\beta\|\mathbf{A}\|_2^2)^{-1}\mathbf{A}^T[\lambda_k + \beta(\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{y}_k - \mathbf{b})] \Big),\\
\mathbf{y}_{k+1} =& \operatorname{Prox}_{(\beta\|\mathbf{B}\|_2^2)^{-1} f_2}\Big(\mathbf{y}_k\\
&- (\beta\|\mathbf{B}\|_2^2)^{-1}\mathbf{B}^T[\lambda_k + \beta(\mathbf{A}\mathbf{x}_{k+1} + \mathbf{B}\mathbf{y}_k - \mathbf{b})] \Big),\\
\lambda_{k+1} =& \lambda_k + \beta(\mathbf{A}\mathbf{x}_{k+1} + \mathbf{B}\mathbf{y}_{k+1} - \mathbf{b}).
\end{aligned}$$

For simplicity we set $\beta = 1$, then the inspired network can be represented as:

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{y}_k - \lambda_k + T_1^k(\mathbf{x}_{k+1}),\\
\mathbf{y}_{k+1} &= \mathbf{x}_{k+1} + \lambda_k + T_2^k(\mathbf{y}_{k+1}),\\
\lambda_{k+1} &= \lambda_k + \mathbf{x}_{k+1} - \mathbf{y}_{k+1}.
\end{aligned}$$

We use a fixed-point iteration method to approximate this implicit network. Specially, we take just one fixed-point iteration to calculate $\mathbf{z}_{k+1}$ from $\mathbf{z}_k$, i.e., we treat $T(\mathbf{z}_{k+1})$ as $T(\mathbf{z}_k)$ when calculating $\mathbf{z}_{k+1}$. From the proof of Theorem 1, it is not hard to see that the universality still holds for the modified network. The architecture of OptDNN-LADMM1 is shown in Figure 11.

- **OptDNN-LADMM2 (Linearized ADMM 2):**

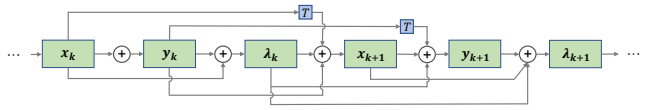


Fig. 11. The architecture of OptDNN-LADMM1.

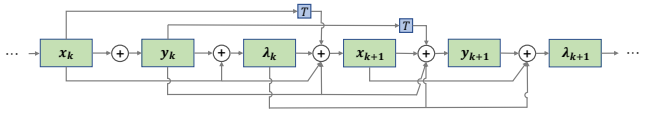


Fig. 12. The architecture of OptDNN-LADMM2.

Write the primal problem as Eq. (77). Another linearized ADMM consists of iterations:

$$\begin{aligned}\mathbf{x}_{k+1} =&\mathbf{x}_k - (L_{f_1} + \beta\|\mathbf{A}\|_2^2)^{-1}\{\nabla f(\mathbf{x}_k)\\ &+ \mathbf{A}^T[\lambda + \beta(\mathbf{A}\mathbf{x}_k) + \mathbf{B}\mathbf{y}_k - \mathbf{b}]\},\\ \mathbf{y}_{k+1} =&\mathbf{y}_k - (L_{f_2} + \beta\|\mathbf{B}\|_2^2)^{-1}\{\nabla g(\mathbf{y}_k)\\ &+ \mathbf{B}^T[\lambda_k + \beta(\mathbf{A}\mathbf{x}_{k+1} + \mathbf{B}\mathbf{y}_k - \mathbf{b})]\},\\ \lambda_{k+1} =&\lambda_k + \mathbf{x}_{k+1} - \mathbf{y}_{k+1},\end{aligned}$$

where L_{f_1} and L_{f_2} are the Lipschitz constant of ∇f_1 and ∇f_2, respectively. Similar to the approach we use in OptDNN-LADMM1, we set $L_{f_1} = L_{f_2} = 1$ and $\beta = 1$ for simplicity, the inspired network can be represented as

$$\begin{aligned}\mathbf{x}_{k+1} &= \frac{1}{2}(\mathbf{x}_k + T_1^k(\mathbf{x}_k) + \mathbf{y}_k - \lambda_k),\\ \mathbf{y}_{k+1} &= \frac{1}{2}(\mathbf{y}_k + T_2^k(\mathbf{y}_k) + \mathbf{x}_{k+1} + \lambda_k),\\ \lambda_{k+1} &= \lambda_k + \mathbf{x}_{k+1} - \mathbf{y}_{k+1}.\end{aligned}$$

The architecture of OptDNN-LADMM2 is shown in Figure 12.

- **ImpOptDNN-PGD (Proximal Gradient Descent):**
The primal problem $\min_{\mathbf{x}} f(\mathbf{x})$ can be equivalently written as

$$\min_{\mathbf{x}} f_1(\mathbf{x}) + f_2(\mathbf{x}),$$

where $f_1(\mathbf{x}) = f_2(\mathbf{x}) = f(\mathbf{x})$.
The proximal gradient descent method consists of iterations:

$$\mathbf{x}_{k+1} = \mathrm{Prox}_{t_k f}(\mathbf{x}_k - t_k\nabla f(\mathbf{x}_k)). \tag{78}$$

The inspired network can be represented as:

$$\begin{aligned}\mathbf{y}_k &= \mathbf{x}_k + T_1^k(\mathbf{x}_k),\\ \mathbf{x}_{k+1} &= \mathbf{y}_k + T_2^k(\mathbf{x}_{k+1}).\end{aligned}$$

- **ImpOptDNN-FISTA (FISTA):**
Write the primal problem as Eq. (78), the FISTA method consists of iterations:

$$\begin{aligned}\mathbf{y}_k &= \mathbf{x}_{k-1} + \frac{k-2}{k+1}(\mathbf{x}_{k-1} - \mathbf{x}_{k-2}),\\ \mathbf{x}_k &= \mathrm{Prox}_{t_k f_2}(\mathbf{y}_k - t_k\nabla f(\mathbf{y}_k)).\end{aligned}$$

The inspired network can be represented as:

$$\begin{aligned}\mathbf{y}_k &= \mathbf{x}_{k-1} + \frac{k-2}{k+1}(\mathbf{x}_{k-1} - \mathbf{x}_{k-2}),\\ \mathbf{z}_k &= \mathbf{y}_k + T_1^k(\mathbf{y}_k),\\ \mathbf{x}_k &= \mathbf{z}_k + T_2^k(\mathbf{x}_k).\end{aligned}$$

- **ImpOptDNN-APG (Accelerated Proximal Gradient Method 2):**
Still writing the primal problem as Eq. (78), utilizing the accelerated proximal gradient method in Eq. (76) to solve this problem, the inspired network can be represented as:

$$\begin{aligned}\mathbf{y}_k &= \theta_k\mathbf{z}_k + (1-\theta_k)\mathbf{x}_k,\\ \mathbf{z}_{k+1} &= \frac{1}{\theta_k}\mathbf{y}_k + \frac{\theta_k - 1}{\theta_k}\mathbf{x}_k + T_1^k(\mathbf{y}_k) + T_2^k(\mathbf{z}_{k+1}),\\ \mathbf{x}_{k+1} &= (1-\theta_k)\mathbf{x}_k + \theta_k\mathbf{z}_{k+1}.\end{aligned}$$

C.2 More details of practical modifications

In this subsection, we will give more details of the practical modifications in Section 5 and show that they do not contradict to our analysis of the universality of OptDNNs.

1) The full connection in the T in Eq. (5) can be transformed into convolution. T can also include normalization layers, such as batch normalization [80] or layer normalization layers [81]. These modifications will not contradict to our analysis of the universality of OptDNN above. It is shown that there is an equivalence between the approximation by FCNNs and by convolution neural networks (CNNs) (Including the ResNet-type CNNs) [82], [83]. Also, the normalization will not hurt the expressive power of the network [80], [81] since the normalization layer can represent the identity map. Besides, CNNs with a special downsampling operation can also behave like FCNNs in approximation [84], thus T can also include downsampling, as well as upsampling.
2) The learnable modules T in Eq. (4) can be of various forms other than $T(\mathbf{z}) = \mathbf{V}\sigma(\mathbf{W}\mathbf{z} + \mathbf{b})$. For example, we can take T as the commonly used 'ReLU before addition' function [24]

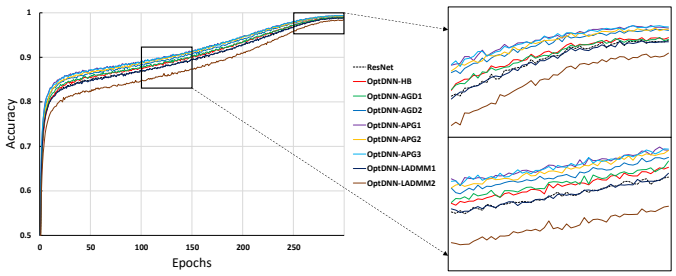
$$\tilde{T}_i^k(\mathbf{z}) = \sigma_R(\mathbf{V}_i^k\sigma_R(\mathbf{W}_i^k\mathbf{z} + \mathbf{b}_{i,1}^k) + \mathbf{b}_{i,2}^k), \tag{79}$$

or the 'full pre-activation' function [24]

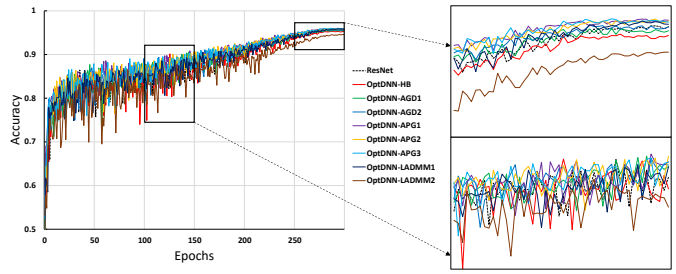
$$\tilde{T}_i^k(\mathbf{z}) = \mathbf{V}_i^k\sigma_R(\mathbf{W}_i^k\sigma_R(\mathbf{z}) + \mathbf{b}_{i,1}^k) + \mathbf{b}_{i,2}^k, \tag{80}$$

where we omit the normalization layers in these two functions. To ensure the universality of the OptDNN, we can follow the idea of Proposition 5 and Theorem 2 to choose appropriate forms of T. Specifically, as long as T satisfies the conditions in Proposition 5, the universality of OptDNNs can still be achieved with the same proof technique. For the full pre-activation function that we use in our experiment, the universality of OptDNNs with this $\tilde{T}$ can be proved more directly: For any T of Eq. (5), it holds that

$$\begin{aligned}T_i^k(\mathbf{z}) &= \mathbf{V}_i^k\sigma_R(\mathbf{W}_i^k\mathbf{z} + \mathbf{b}_i^k)\\ &= \mathbf{V}_i^k\sigma_R(\mathbf{W}_i^k\sigma_R(\mathbf{z} + \mathbf{b}_{i,2}^{k-1}) - \mathbf{W}_i^k\mathbf{b}_{i,2}^{k-1} + \mathbf{b}_{i,1}^k),\end{aligned}$$

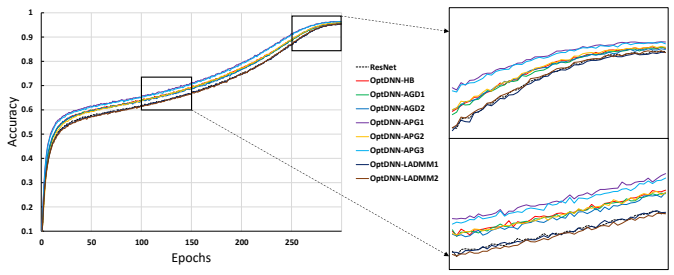


(a) Training accuracy on CIFAR-10 in the shallower and wider setting.

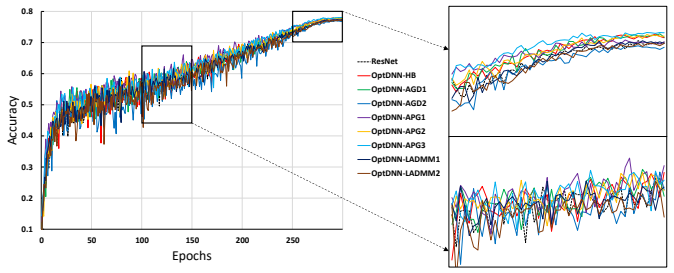


(b) Test accuracy on CIFAR-10 in the shallower and wider setting.

Fig. 13. The dynamics of training and test accuracy on CIFAR-10.



(a) Training accuracy on CIFAR-100 in the deeper and narrower setting.



(b) Test accuracy on CIFAR-100 in the deeper and narrower setting.

Fig. 14. The dynamics of training and test accuracy on CIFAR-100.

where $\mathbf{b}_{i,2}^{k-1}$ is a vector that satisfies $\mathbf{z}+\mathbf{b}_{i,2}^{k-1}>0$. Thus T_i^k can be realized by $\tilde{T}_i^k$.

3) The network can be divided into different stages, where the formulations in OptDNN are used in each stage and the size of feature maps may change between the stages. This can reduce the number of parameters and improve training efficiency. If the depth of each stage of OptDNN can be arbitrarily large, the universality will still hold for the network. This can be understood from composite approximation, i.e., for a composite function $g = f_1 \circ f_2 \circ \cdots \circ f_r$, if we can approximate each component function f_i to any precision by continuous function on a compact domain, we can approximate g arbitrarily well [85]. Then we can prove the universality of OptDNN from Proposition 1 by first approximating the target function from a continuous composite function and then approximating each component function by OptDNN.
4) The modulating coefficients in the network can also be set learnable. From the perspective of approximation, OptDNNs with learnable modulating coefficients are more expressive than those with fixed modulating coefficients. Our proof naturally holds in the setting of learnable modulating coefficients.

C.3 More implementation details of OptDNNs

In this subsection, we will give more implementation details of OptDNNs in our image classification experiment.

For experiments on CIFAR datasets, our OptDNNs first go through a convolution module to increase the channel dimension to the base width. Then we stack multiple blocks corresponding to the iterations of the optimization algorithm. The parameterized function in each block consists of BN-ReLU-Conv-BN-ReLU-Conv. Since the receptive field is important for convolution operations, we will reshape the data by a squeeze operation between several blocks. Specifically, for a $C \times H \times W$ tensor, the squeeze operation unfolds each 2×2 spatial patch into the channel dimension, resulting in a $4C \times \frac{H}{2} \times \frac{W}{2}$ tensor. This operation can enlarge the receptive field for successive convolution operations and also increase the channel dimension. For our shallower and wider setting, the base width is set as 32 and there will be 2 squeeze operations, so the final channel dimension is 512. Block numbers for different stages are 2, 4, and 2. For our deeper and narrower setting, the base width is set as 16 and there will be 1 squeeze operation, so the final channel dimension is 64. Block numbers for different stages are both 22. As for OptDNN-AGD2, OptDNN-APG1, and OptDNN-APG3, which require connections from all previous states, since parameters grow quadratically with depth, we restart the dense connection after each squeeze operation. This can be viewed as a composite of multiple OptDNNs. The block number for different stages are 2, 2, 2, and 8, 6 for the shallower and wider and the deeper and narrower settings, respectively, and the base width for the shallower and wider setting is set as 27 to keep a similar amount of parameters. As for OptDNN-LADMM1 and OptDNN-LADMM2, since there are two parameterized functions in each block, we reduce the block number by half to keep the same amount of parameters. For the coefficients in OptDNNs such as β or $\frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}$, we set them to be learnable and we initialize them with the original value. For OptDNN-HB, β is initialized as 1. For OptDNN-AGD2 and OptDNN-APG1, the parameter $h_{k+1,j}$ before the parameterized function T is integrated into T. For OptDNN-APG2, the coefficients such as θ_k and $\frac{2}{1+\theta_k}$ are separate learnable parameters for the calculation of $\mathbf{y}_k$, $\mathbf{z}_{k+1}$, and $\mathbf{x}_{k+1}$, i.e. the expressions are

$$\begin{aligned}\mathbf{y}_k &= \theta_1\mathbf{z}_k + (1-\theta_1)\mathbf{x}_k,\\ \mathbf{z}_{k+1} &= \theta_2\mathbf{y}_k + (1-\theta_2)\mathbf{x}_k + T^k(\mathbf{y}_k),\\ \mathbf{x}_{k+1} &= \theta_3\mathbf{x}_k + (1-\theta_3)\mathbf{z}_{k+1},\end{aligned}$$

where $\theta_1, \theta_2, \theta_3$ are learnable parameters initialized corre-

sponding to the original value. For OptDNN-APG3, coefficients are separate learnable parameters for $\mathbf{x}_0$ and $\mathbf{y}_i$. Experiments for CIFAR-10 and CIFAR-100 are based on 3 runs under the same random seed 2022, 0, and 1.

For implicit models, BN is replaced with GN in each block and we solve the implicit equation by unrolling the fixed-point update for 5 iterations.

As for implementation in similar settings as ResNet and DenseNet, we replace each block of ResNet or DenseNet with OptDNN blocks as introduced above, and we keep other settings the same, except that the functions f of DenseNets act on the concatenated $[\mathbf{x}_1, \cdots, \mathbf{x}_k]$ while the functions f of OptDNNs act on separate $\mathbf{x}_i$ and are then integrated. OptDNN-HB, OptDNN-AGD1, OptDNN-APG2, OptDNN-LADMM1, and OptDNN-LADMM2 are realized in the ResNet-like setting, while OptDNN-AGD2, OptDNN-APG1, and OptDNN-APG3 are formulated in the DenseNet-like setting since they have dense connections from previous steps. For OptDNN3D in the DenseNet-like setting, we integrate the residual term $\mathbf{x}_k$ into $T(\mathbf{x}_k)$.

For experiments on ImageNet, we follow the same settings as ConvNeXt, ConvNeXt (iso.), and ViT, respectively. We replace each residual block in their model with OptDNN-APG2 blocks and keep the learnable modules and other settings the same. The experiments are carried out under the default random seed of ConvNeXt and ViT.

C.4 Additional Figures.

We provide the dynamics of the training accuracy and test accuracy of different models on CIFAR datasets in Figures 13 and 14.

The evolution of data points as the depths of networks increase for all explicit OptDNNs in the nested rings separation task is provided in Figure 15.

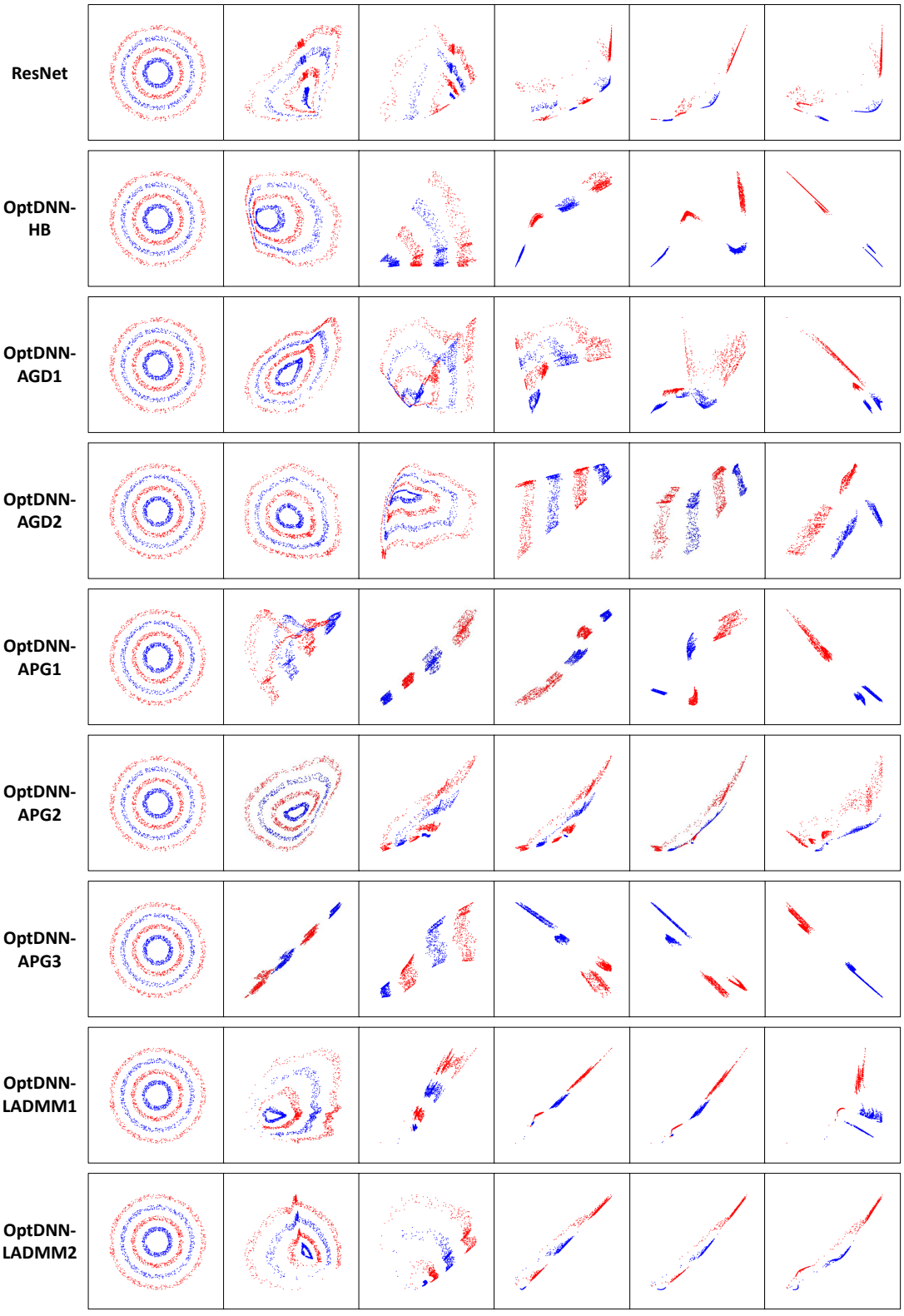


Fig. 15. Evolution of data points as the depths of networks increase for OptDNNs in the nested rings separation task. From left to right, each figure represents data points transformed at layer $3k$.