Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2020-0214

Vol. **28**, No. 5, pp. 1838-1885
November 2020

Butterfly-Net: Optimal Function Representation Based on Convolutional Neural Networks

Yingzhou Li[1], Xiuyuan Cheng[1,*] and Jianfeng Lu[1,2]

[1] *Department of Mathematics, Duke University, Durham, NC 27708, USA.*
[2] *Department of Chemistry and Department of Physics, Duke University, Durham, NC 27708, USA.*

Received 30 October 2020; Accepted 6 November 2020

Abstract. Deep networks, especially convolutional neural networks (CNNs), have been successfully applied in various areas of machine learning as well as to challenging problems in other scientific and engineering fields. This paper introduces *Butterfly-net*, a low-complexity CNN with structured and sparse cross-channel connections, together with a *Butterfly* initialization strategy for a family of networks. Theoretical analysis of the approximation power of *Butterfly-net* to the Fourier representation of input data shows that the error decays exponentially as the depth increases. Combining *Butterfly-net* with a fully connected neural network, a large class of problems are proved to be well approximated with network complexity depending on the effective frequency bandwidth instead of the input dimension. Regular CNN is covered as a special case in our analysis. Numerical experiments validate the analytical results on the approximation of Fourier kernels and energy functionals of Poisson's equations. Moreover, all experiments support that training from *Butterfly* initialization outperforms training from random initialization. Also, adding the remaining cross-channel connections, although significantly increases the parameter number, does not much improve the post-training accuracy and is more sensitive to data distribution.

AMS subject classifications: 15A23, 65D05, 65F10, 62G08, 68W20, 68W25

Key words: Butterfly algorithm, convolutional neural network, Fourier analysis, deep learning.

1 Introduction

Deep neural network is a central tool in machine learning and data analysis nowadays [5]. In particular, convolutional neural network (CNN) has been proved to be a powerful tool

*Corresponding author. *Email addresses:* yingzhou.li@duke.edu (Y. Li), xiuyuan.cheng@duke.edu (X. Cheng), jianfeng@math.duke.edu (J. Lu)

in image recognition and representation. Deep learning has also emerged to be successfully applied in solving PDEs [6, 28, 37] and physics problems [4, 17, 35, 47, 53], showing the potential of becoming a tool of great use for computational mathematics and physics as well. Given the wide application of PDEs and wavelet based methods in image and signal processing [8, 11, 39], an understanding of CNN's ability to approximate differential and integral operators will lead to an explanation of CNN's success in these fields, as well as possible improved network architectures.

The remarkable performance of deep neural networks across various fields relies on their ability to accurately represent functions of high-dimensional input data. Approximation analysis has been a central topic to the understanding of the neural networks. The classical theory developed in 80's and early 90's [3, 13, 26] approximates a target function by a linear combination of sigmoids, which is equivalent to a fully connected neural network with one hidden layer. While universal approximation theorems were established for such shallow networks, the research interest in neural networks only revived in recent years after observing the successful applications of deep neural networks, particular the superior performance of CNNs in image and signal processing.

Motivated by the empirical success, the approximation advantage of deep neural networks over shallow ones has been theoretically analyzed in several places. However, most results assume stacked fully connected layers and do not apply to CNNs which have specific geometrical constraints: (1) the convolutional scheme, namely local-supported filters with weight sharing, and (2) the hierarchical multi-scale architecture. The approximation power of deep networks with hierarchical geometrically-constrained structure has been studied recently [12,40,41], yet the network architecture differ from the regular CNN. The approximation theory of CNN has been studied in [2, 54]. We review the related literature in more detail below.

This paper proposes a specific architecture under the CNN framework based on the *Butterfly* scheme originally developed for the fast computation of special function transforms [42, 44, 52] and Fourier integral operators [9, 10, 31–34]. *Butterfly* scheme provides a hierarchical structure with locally low-rank interpolation of kernel functions and can be applied to solve many PDE related problems. In terms of computational complexity, the scheme is near optimal for Fourier kernels and Fourier integral operators. The proposed *Butterfly-net* explicitly adopts the hierarchical structure in *Butterfly* scheme as the stacked convolutional layers. If the parameters are hard-coded as that in the *Butterfly* scheme (*Butterfly* initialization), then *Butterfly-net* collectively computes the Fourier coefficients of the input signal with guaranteed numerical accuracy. Unlike regular CNN which has dense cross-channel connections, the channels in the *Butterfly-net* have clear correspondences to the frequency bands, namely the position in the spectral representation of the signal, and meanwhile, the cross-channel weights are sparsely connected. In this paper, we also study *Butterfly-net* with dense cross-channel connections, which is named *Inflated-Butterfly-net*. Regular CNN is a special *Inflated-Butterfly-net* [49]. Comparing *Butterfly-net* and *Inflated-Butterfly-net*, *Butterfly-net* is much lighter: the model complexity (in terms of parameter number) is $\mathcal{O}(K\log N)$ and computational complexity is

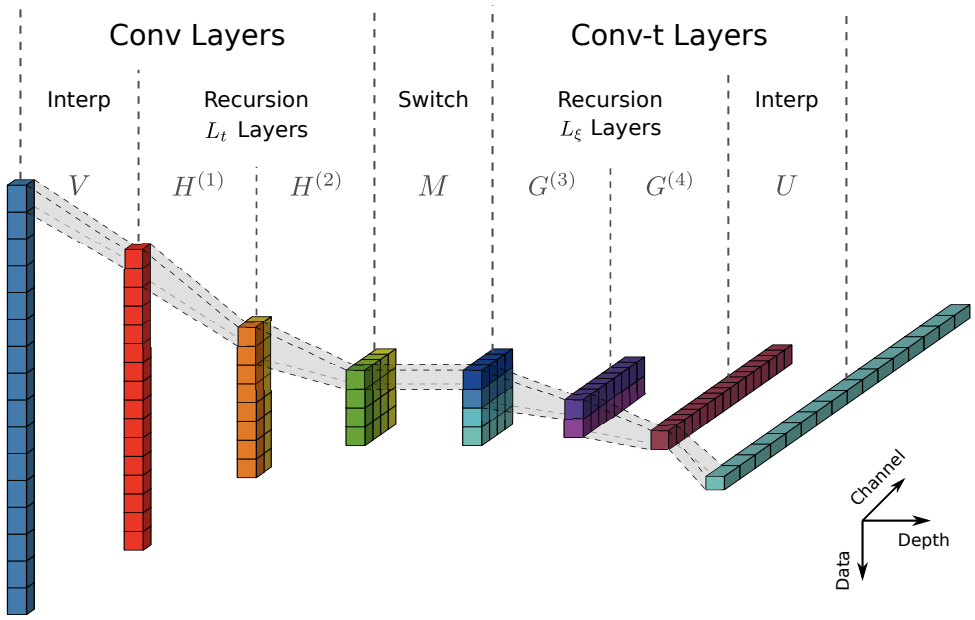


Figure 1: Illustration of a *Butterfly-net*: The network consists of 3 modules, (1) Conv layers, which has one Interpolation later and L_t Recursion layers, (2) a Switch layer, and (3) Conv-t layers, which has $L_{\tilde{\xi}}$ Recursion layers and one Interpolation later. The conv and conv-t layers have convolutional and transpose convolutional structure respectively, and the Switch layer only involves local operations. Detailed structures of these layers are given in Section 3. This figure shows the case when $L_t=L_{\tilde{\xi}}=2$, and the notations $V,H^{(\ell)},M,G^{(\ell)},U$ denote the matrix representations of the operations in the scheme, c.f. Section 3.4. The name *"butterfly"* can be understood from the shape of the network.

$\mathcal{O}(N\log N)$, where N is the length of the discrete input signal and K is the frequency bandwidth.

The approximation error of *Butterfly-net* in representing Fourier kernels is proved to exponentially decay as the network depth increases, which is numerically validated in Section 5. Due to the efficient approximation of Fourier kernels, *Butterfly-net* thus possesses all approximation properties of the Fourier representation of input signals, which is particularly useful for solving PDEs and (local) Fourier-based algorithms in image and signal processing. Theoretically, the approximation guarantee for *Butterfly-net* to represent Fourier kernels leads to an approximation result of a family of sparsified CNNs and regular CNNs. The informal statement of our main result on neural network function approximation is as follows:

Theorem 1.1 (Informal version)**.** *Consider using neural network to approximate a function $f(\vec{x})$, $\vec{x}\in\mathcal{X}\subset\mathbb{R}^N$, where f can be approximated by a band-limited function of $\vec{x}$ with bandwidth K (this may be due to the frequency decay of f, or $\mathcal{X}$, or both). Then there exists a family of Butterfly-nets whose numbers of parameters are all bounded by*

$$n_b \lesssim K\log N, \tag{1.1}$$

such that using anyone of them to extract a deep feature of length K from the input x can reduce the effective dimension of the input data from N to K for neural network function approximation.

The dimension reduction is reflected by the needed network complexity in approximating $f(\vec{x})$ by a fully-connected network, namely a reduction from a factor of $\varepsilon^{-N/s}$ (for a fully connected net to approximate $f(\vec{x})$ directly) to that of $\varepsilon^{-K/s}$ (for a fully connected net to approximate g, *Butterfly-net* to approximate $\mathcal{B}$, and $f(\vec{x}) \approx g(\mathcal{B}\vec{x})$), where s is the regularity level and ε is the uniform approximation error of f. The precise statement is given in Theorem 4.2.

Moreover, in the above statement, the family of *Butterfly-net*s can be replaced by the corresponding family of *Inflated-Butterfly-net* with $n_b \lesssim K^2 \log N$ parameters and the same dimension reduction argument still holds. In particular, one member in the family of *Inflated-Butterfly-net*s is a regular CNN. Hence, our approximation analysis covers regular CNNs as a special case.

Our methodology and theoretical results can be generalized to signal $\vec{x}$ lying on a d-dimensional grid, e.g., image data on a 2D grid. In this case, the *Butterfly-net* is a 2D CNN (with sparse cross-channel connections) that provably approximates the kernel of 2D Fourier Transform. The connection and extension will be discussed later, see comments beneath Theorem 2.1 and Remark 4.5.

1.1 Contributions

Our contributions can be summarized as follows.

1) We propose a family of novel neural network architectures, named *Butterfly-net*s, which are composed of convolutional and transpose convolutional layers with sparse cross-channel connections, plus a locally connected switch layer in between. An associated *Butterfly* initialization strategy is proposed for *Butterfly-net*s to approximate Fourier kernels. The *Butterfly-net* architecture can be inflated via replacing the sparse cross-channel connections by dense connections, and this contains regular CNN as a special case.

2) The approximation error of *Butterfly-net*s representing the Fourier kernels is theoretically proved to be exponentially decay as the network depth increases. Concatenating *Butterfly-net* (or its inflated version) with a fully-connected layer, we provide an approximation analysis for a wide class of functions with frequency decay property and the approximation complexity depends on the effective dimension K instead of the input data dimension N. The regular CNN is covered as a special case.

3) Numerically, we apply *Butterfly-net* and its inflated version to a wide range of datasets. The successful trainings on all datasets support our approximation analysis. Further, we find that training from *Butterfly* initialization in all cases outperforms training from random initialization, especially when the output function only depends on a few frequencies among the wide frequency band of the input data. *Butterfly-net* achieves

similar post-training accuracy as its inflated version with far less number of parameters. *Butterfly-net* also admits better transfer learning capability when the distribution of the testing data is shifted away from the training data.

1.2 Related works

Before we explain more details in the rest of the paper, we review some related works.

Fast algorithm inspired neural network structures. Fast algorithm has inspired several neural network structures recently. Based on $\mathcal{H}$-matrix and $\mathcal{H}^2$-matrix structure, Fan and his coauthors proposed two multiscale neural networks [20, 21], which are more suitable in training smooth linear or nonlinear operators due to the multiscale nature of $\mathcal{H}$-matrices. In addition to that, nonstandard wavelet form inspired the design of BCR-Net [22], which is applied to address the inverse of elliptic operator and nonlinear homogenization problem and recently been embedded in a neural network for solving electrical impedance tomography [19] and pseudo-differential operator [23]. Multigrid method also inspired MgNet [25]. In addition to the above approximation of relatively smooth operators, *Butterfly* scheme inspired the design of SwitchNet [27], which is a non-convolutional three layer neural network and addresses scattering problems.

Classical approximation results of neural networks. Universal approximation theorems for fully-connected neural networks with one hidden layer were established in [13, 26] showing that such networks can approximate a target function with arbitrary accuracy if the hidden layer is allowed to be wide enough. In theory, the family of target functions can include all measurable functions [26], when exponentially many hidden neurons are used. Gallant and White [24] proposed "Fourier network", proving universal approximation to squared-integrable functions by firstly constructing a Fourier series approximation of the target function in a hard-coded way. These theorems are firstly proved for one-dimensional input, and when generalizing to the multivariate case the complexity grows exponentially.

Using the Fourier representation of the target function supported on a sphere in $\mathbb{R}^d$, Barron [3] showed that the mean squared error of the approximation, integrated with arbitrary data distribution on the sphere, decays as $\mathcal{O}(n^{-1})$ when n hidden nodes are used in the single hidden layer. The results for shallow networks are limited, and the approximation power of depth in neural networks has been advocated in several recent works, see below. Besides, while the connection to Fourier analysis was leveraged, at least in [3,24], it is different from the hierarchical function representation scheme as what we consider here.

Approximation power of deep neural networks. The expressive power of deep neural networks has drawn many research interests in recent years. The approximation power of multi-layer restricted Boltzmann machines (RBM) was studied in [30], which showed that RBMs are universal approximators of discrete distributions and more hidden layers

improves the approximation power. Relating to the classical approximation results in harmonic analysis, Bölcskei et al. [7] derived lower bounds for the uniform approximation of square-integrable functions, and proved the asymptotic optimality of the sparsely connected deep neural networks as a universal approximator. However, the network complexity also grows exponentially when the input dimension increases.

The approximation advantage of deep architecture over shallow ones has been studied in several works. Delalleau and Bengio [14] identified a deep sum-product network which can only be approximated by an exponentially larger number of shallow ones. The exponential growth of linear regions as the number of layers increases was studied in [43, 48]. Eldan and Shamir [18] constructed a concrete target function which distinguishes three and two-layer networks. Liang and Srikant [36] showed that shallow networks require exponentially more neurons than deep networks to obtain a given approximation error for a large class of functions. The advantage of deep ReLU networks over the standard single-layer ones was analyzed in [51] in the context of approximation in Sobolev spaces. Lu et al. [38] shows the advantage of deep ReLU networks in approximating smooth (band-limited) functions. The above works address deep networks with fully-connected layers, instead of having geometrically-constrained constructions like CNNs.

Deep neural networks with such geometric constraints are relatively less analyzed. The approximation power of a hierarchical binary tree network was studied in [40, 41] which supports the potential advantage of deep CNNs. Cohen et al. [12] used convolutional arithmetic circuits to show the equivalence between a deep network and a hierarchical Tucker decomposition of tensors, and proved the advantage of depth in function approximation. The networks being studied differ from the regular CNNs widely used in the typical real world applications. Recently, Zhou [54] proposed the universal approximation theory of deep CNN with an estimation on the number of free parameters. Bao et al. [2] shows the approximation power of CNNs over deep neural networks without constraints on a class of functions. Comparing to [54], our analysis covers a wider range of networks and the approximation complexity is much lower on a restricted function class. Comparing to [2], different function classes are discussed and we both show the advantage of CNN.

1.3 Organization

The rest of this paper is organized as follows. Building on top of the traditional *Butterfly* literature, Section 2 first shows the low-rank property of Fourier kernel and illustrates the *Butterfly* algorithm tailored for Fourier kernel. Section 3 proposes interpolative convolutional layer as building blocks for *Butterfly-net* followed by the *Butterfly-net* architecture with *Butterfly* initialization and its matrix representation. Section 4 analyzes the approximation power of *Butterfly-net* on Fourier kernels and its extension to general functionals. Numerical results of *Butterfly-net* and *Inflated-Butterfly-net* are presented in Section 5 including the performance comparison of network architectures, initializations,

and datasets. Finally, in Section 6, we conclude the paper together with discussion on future directions.

2 Preliminaries

This section paves the path to *Butterfly-net*. We first derives a low-rank interpolation of the Fourier kernel, which is crucial to the efficiency of *Butterfly* scheme and *Butterfly-net*. Then in Section 2.2, we reviews the *Butterfly* algorithm for Fourier kernel. Readers, who are familiar with butterfly scheme, should be safe to skip it.

2.1 Low-rank approximation of Fourier kernel

Fourier kernel throughout this paper is defined as

$$\mathcal{K}(\xi,t)=e^{-2\pi\imath\xi\cdot t},\qquad \xi\in[K_0,K_0+K),\quad t\in[0,1), \tag{2.1}$$

where $[K_0,K_0+K)$ denotes the frequency window of interests, K_0 denotes the starting frequency, and K denotes the frequency window width. It is well-known that the discrete Fourier transform (DFT) matrix, i.e., uniform discretization of (2.1) with proper scaling, has orthonormal rows and columns. Hence, the DFT matrix is a unitary matrix of full rank and the Fourier kernel is also full rank. Theorem 2.1 show when the Fourier kernel is restricted to certain pairs of subdomains of $[0,1)$ and $[K_0,K_0+K)$, it has low-rank property.

We first give a brief introduction of the Chebyshev interpolation with r points. The Chebyshev grid of order r on $[-\frac{1}{2},\frac{1}{2}]$ is defined as

$$\left\{z_i=\frac{1}{2}\cos\left(\frac{(i-1)\pi}{r}\right)\right\}_{i=1}^{r}. \tag{2.2}$$

The Chebyshev interpolation of a function $f(x)$ on $[-\frac{1}{2},\frac{1}{2}]$ is defined as

$$\Pi_r f(x)=\sum_{k=1}^{r} f(z_k)\mathcal{L}_k(x), \tag{2.3}$$

where $\mathcal{L}_k(x)$ is the Lagrange polynomial as

$$\mathcal{L}_k(x)=\prod_{p\neq k}\frac{x-z_p}{z_k-z_p}. \tag{2.4}$$

Several earlier works [9, 10, 33] proved the Chebyshev interpolation representation for Fourier integral operators, which are generalized Fourier kernel. Theorem 2.1 is a special case of these earlier work but with more precise and explicit estimation on the prefactor.

Theorem 2.1. *Let r be the number of Chebyshev points, $B\subset[0,1)$ and $A\subset[K_0,K_0+K)$ denote a domain pair such that $w(A)w(B)\leq\frac{r}{\pi e}$, where $w(\cdot)$ is the domain length function. Then there exists low-rank representations of the Fourier kernel restricted to the domain pair*

$$\sup_{\xi\in A,t\in B}\left|e^{-2\pi\imath\xi\cdot t}-\sum_{k=1}^{r}e^{-2\pi\imath\xi\cdot t_k}e^{-2\pi\imath\xi_0\cdot(t-t_k)}\mathcal{L}_k(t)\right|\leq\left(2+\frac{2}{\pi}\ln r\right)\left(\frac{\pi ew(A)w(B)}{2r}\right)^r, \quad (2.5)$$

and

$$\sup_{\xi\in A,t\in B}\left|e^{-2\pi\imath\xi\cdot t}-\sum_{k=1}^{r}e^{-2\pi\imath(\xi-\xi_k)\cdot t_0}\mathcal{L}_k(\xi)e^{-2\pi\imath\xi_k\cdot t}\right|\leq\left(2+\frac{2}{\pi}\ln r\right)\left(\frac{\pi ew(A)w(B)}{2r}\right)^r, \quad (2.6)$$

where ξ_0 and t_0 are the centers of A and B, and ξ_k and t_k are Chebyshev points on A and B.

The proof of Theorem 2.1 is deferred to Appendix A. This property is also known as complementary low-rank property [32, 33]. In [33], an analogous theorem is proved for d-dimensional Fourier integral operators, where a special corona domain partition on the frequency domain is introduced to handle the singularity near the origin. Fortunately for d-dimensional Fourier kernels, there is no such singularity. As a result, the extension to d-dimensional Fourier kernel can be simplified from the analysis in [33]. Thus, by constructing a d-dimensional Chebyshev interpolation as a tensor product of one dimensional Chebyshev interpolations, one can adopt the same proof of Theorem 2.1 in Appendix A and obtain an analog approximation bound as in Theorem 2.1 for d-dimensional Fourier kernel. The analog approximation bound has the same growth factor as that in Theorem 2.1 for r being the Chebyshev points on each dimension, while the prefactor is bounded by $\left(2+\frac{2}{\pi}\ln r\right)^d$.

2.2 Butterfly algorithm for Fourier kernel

This section briefly describes the *Butterfly* algorithm tailored for Fourier kernel based on Theorem 2.1. Given a function $x(t)$ discretized on a uniform grid, $\{\bar{t}_q=\frac{q-1}{N}\}_{q=1}^N$, the goal is to compute the discrete Fourier transform $\{\hat{x}(\bar{\xi}_p),\ \bar{\xi}_p=K_0,\cdots,K_0+K-1\}$ defined by

$$\hat{x}(\bar{\xi}_p)=\sum_{\bar{t}_q}e^{-2\pi\imath\bar{\xi}_p\bar{t}_q}x(\bar{t}_q),\quad \bar{\xi}_p=K_0,\cdots,K_0+K-1. \quad (2.7)$$

Without loss of generality, we assume $N\geq K$ throughout this paper.

Hierarchical domain partition. In order to benefit from Theorem 2.1, we first define the L layer hierarchical partition of $[0,1)$ for $L\leq\log N$. [†] Extension to $L>\log N$ is possible but not common in butterfly scheme. Let $B_0^0=[0,1)$ be the domain on layer 0. On layer 1, the

[†]Without further explanation, log denotes logarithm base 2 and ln denotes natural logarithm base e.

Table 1: Complementary domain pairs used in *Butterfly-net* on all layers. $\sharp\{i\}$ and $\sharp\{j\}$ are lengths of index i and j, $w(\cdot)$ is the domain length function, and $L_{\min}=\min(L_t,\log K-L_\xi)$.

	Frequency			Time			
ℓ range	domain	$\sharp\{i\}$	$w(A)$	domain	$\sharp\{j\}$	$w(B)$	$w(A)w(B)$
$0\le\ell\le L_{\min}$	A_i^ℓ	2^ℓ	$K\cdot 2^{-\ell}$	$B_j^{L-\ell}$	$2^{L-\ell}$	$2^{\ell-L}$	$K\cdot 2^{-L}$
$L_{\min}<\ell\le L_t$	A_i^ℓ	$2^{L_{\min}}$	$K\cdot 2^{-L_{\min}}$	$B_j^{L-\ell}$	$2^{L-\ell}$	$2^{\ell-L}$	$K\cdot 2^{\ell-L_{\min}-L}$
$L_t<\ell\le L$	A_i^ℓ	$2^{\ell-L_t+L_{\min}}$	$K\cdot 2^{-\ell+L_t-L_{\min}}$	$B_j^{L-\ell}$	$2^{L-\ell}$	$2^{\ell-L}$	$K\cdot 2^{-L_\xi-L_{\min}}$

domain B_0^0 is evenly partitioned into $B_0^1=[0,\frac{1}{2})$ and $B_1^1=[\frac{1}{2},1)$. We conduct the partition recursively, i.e., $B_j^{\ell-1}$ is evenly partitioned into B_{2j}^ℓ and B_{2j+1}^ℓ. In the end, the partition on layer ℓ is $\{B_j^\ell,\ j=0,\cdots,2^\ell-1\}$ and each $B_j^\ell=[\frac{j}{2^\ell},\frac{j+1}{2^\ell})$ is of length $2^{-\ell}$.

Before partitioning the frequency domain, we introduce two more notations, L_t and L_ξ, which split the L layers into two group, i.e.,

$$L=L_t+L_\xi. \tag{2.8}$$

The split of L into L_t and L_ξ will used in Section 3.2 and later analysis. We will apply (2.5) for the first L_t layers to compress the kernel and apply (2.6) for the later L_ξ layers. L_ξ satisfies the constraint $0\le L_\xi\le\log K$.

The partitioning of the frequency domain is described as follows. For layers $\ell\le L_{\min}=\min(L_t,\log K-L_\xi)$, we partition the frequency domain in the same way as for the time domain. Hence the partition is $\{A_i^\ell,\ i=0,\cdots,2^\ell-1\}$ and each $A_i^\ell=[K_0+\frac{i}{2^\ell}K,K_0+\frac{i+1}{2^\ell}K)$ is of length $K\cdot 2^{-\ell}$. For layers $L_{\min}<\ell\le L_t$, the partition remains the same as that on layer $L_{\min}$. The partition is $\{A_i^\ell,\ i=0,\cdots,2^{L_{\min}}-1\}$ and each $A_i^\ell=[K_0+\frac{i}{2^{L_{\min}}}K,K_0+\frac{i+1}{2^{L_{\min}}}K)$ is of length $K\cdot 2^{-L_{\min}}$. For the rest layers $L_t<\ell\le L$, the hierarchical bi-partition is applied again starting from domains on layer $L_{\min}$. The partition is $\{A_i^\ell,\ i=0,\cdots,2^{\ell-L_t+L_{\min}}-1\}$ and each $A_i^\ell=[K_0+\frac{i}{2^{\ell-L_t+L_{\min}}}K,K_0+\frac{i+1}{2^{\ell-L_t+L_{\min}}}K)$ is of length $K\cdot 2^{-\ell+L_t-L_{\min}}$.

Table 1 lists all domain pairs used in *Butterfly* algorithm and *Butterfly-net*, which is important to the later complexity analysis. Theorem 2.1 is applied to all these domain pairs in our later analysis. Notice that when $L\le\log K$, we have $L_{\min}=L_t$ and the second range of ℓ in Table 1 is empty.

Butterfly algorithm. For properly chosen r, the Fourier kernel restricted to each domain pair in Table 1 admits the low-rank representation and hence the submatrix $\{e^{-2\pi i\bar{\xi}_p\bar{t}_q}\}_{\bar{\xi}_p\in A_i^\ell,\bar{t}_q\in B_j^{L-\ell}}$ is approximately of rank r. An explicit formula for the low-rank approximation is given by a discrete version of Theorem 2.1. Then the *Butterfly* algorithm for Fourier kernel can be described layer by layer as follows.

1. *Interpolation* ($\ell=0$). For $A=A_0^0$ and each subdomain $B=B_j^L$, conduct a coefficient transference from uniform grid in B to Chebyshev points in B and denote the trans-

ferred coefficients as $\{\lambda_k^{AB}\}_{1\le k\le r}$, i.e.,

$$\lambda_k^{AB}=\sum_{t\in B}e^{-2\pi\iota\xi_0\cdot(t-t_k)}\mathcal{L}_k(t)x(t),\quad 1\le k\le r, \tag{2.9}$$

where ξ_0 is the center of A and t_k denotes the Chebyshev point. According to Theorem 2.1, we note that

$$\hat{x}(\bar{\xi}_p)=\sum_{B=\{B_j^L\}}\sum_{\bar{t}_q\in B}e^{-2\pi\iota\bar{\xi}_p\cdot\bar{t}_q}x(\bar{t}_q)\approx\sum_{B=\{B_j^L\}}\sum_{t_k\in B}e^{-2\pi\iota\bar{\xi}_p\cdot t_k}\lambda_k^{AB},\quad \bar{\xi}_p\in A, \tag{2.10}$$

where $\bar{t}_q$ and $\bar{\xi}_p$ denote uniform grid points and the approximation accuracy is controlled by r and L based on (2.5).

2. *Recursion* (ℓ=1,$\cdots$,L_t). For each domain pair $(A,B)=(A_i^\ell,B_j^{L-\ell})$, construct the transferred coefficients $\{\lambda_k^{AB}\}_{1\le k\le r}$. Let P denote the parent of A and C denote a child of B at layer $\ell-1$. Throughout, we shall use the notation $C\succ B$ when C is a child of B. At layer $\ell-1$, the coefficients $\{\lambda_s^{PC}\}_{1\le s\le r}$ satisfy

$$\hat{x}(\bar{\xi}_p)\approx\sum_{C=\{B_j^{L-\ell+1}\}}\sum_{t_s^c\in C}e^{-2\pi\iota\bar{\xi}_p\cdot t_s^c}\lambda_s^{PC}=\sum_{B=\{B_j^{L-\ell}\}}\sum_{\substack{C\succ B\\ t_s^c\in C}}e^{-2\pi\iota\bar{\xi}_p\cdot t_s^c}\lambda_s^{PC},\quad \bar{\xi}_p\in P. \tag{2.11}$$

Since $A\subset P$, the above approximation holds for $\bar{\xi}_p\in A$ as well. Now conduct a coefficient transference from Chebyshev points in $C\succ B$ to Chebyshev points in B and denote the transferred coefficients as $\{\lambda_k^{AB}\}_{1\le k\le r}$, i.e.,

$$\lambda_k^{AB}=\sum_{\substack{C\succ B\\ t_s^c\in C}}e^{-2\pi\iota\xi_0\cdot(t_s^c-t_k)}\mathcal{L}_k(t_s)\lambda_s^{PC},\quad 1\le k\le r, \tag{2.12}$$

where ξ_0 denotes the center of A, t_s^c and t_k denote the Chebyshev in C and B respectively. According to Theorem 2.1, the transferred coefficients admit the approximation

$$\hat{x}(\bar{\xi}_p)\approx\sum_{B=\{B_j^{L-\ell}\}}\sum_{t_k\in B}e^{-2\pi\iota\bar{\xi}_p\cdot t_k}\lambda_k^{AB},\quad \bar{\xi}_p\in A. \tag{2.13}$$

3. *Switch* ($\ell=L_t$). For the layer visited $\ell\le L_t$, the Chebyshev interpolation is applied to variable t, while for layer $\ell>L_t$ the interpolation is applied to variable ξ. Hence, we switch the role of t and ξ at this step. For all pairs $(A,B)=(A_i^{L_t},B_j^{L_\xi})$ in the last step, λ_s^{AB} denotes the coefficients obtained by Chebyshev interpolation. Let $\{\xi_k^A\}_k$ and $\{t_s^B\}_s$ denote the Chebyshev points in A and B respectively. Then we abuse notation λ_k^{AB} and define Fourier transformed coefficients for (A,B) as

$$\lambda_k^{AB}:=\sum_{s=1}^{r}e^{-2\pi\iota\xi_k^A\cdot t_s^B}\lambda_s^{AB}\approx\sum_{\bar{t}_q\in B}e^{-2\pi\iota\xi_k^A\cdot\bar{t}_q}x(\bar{t}_q), \tag{2.14}$$

where $\bar{t}_q$ denotes the original uniform distributed points in B and the approximation is due to the definition of λ_s^{AB} and (2.6).

4. *Recursion* $(\ell = L_t+1,\cdots,L_t+L_{\xi})$. For each pair $(A,B)=(A_i^{\ell},B_j^{L-\ell})$, and the corresponding parent domain P and child domain C of A and B respectively, the coefficients $\{\lambda_s^{PC}\}_{1\le s\le r}$ satisfy

$$\lambda_s^{PC} \approx \sum_{\bar{t}_q\in C} e^{-2\pi\imath\xi_s\cdot\bar{t}_q} x(\bar{t}_q), \tag{2.15}$$

where ξ_s denotes the Chebyshev points in P. Given the second approximation in Theorem 2.1, we have, with notation $\bar{\xi}_p$ and ξ_k being uniform points and Chebyshev points in A respectively,

$$\begin{aligned}
&\hat{x}(\bar{\xi}_p)\\
&= \sum_{B=\{B_j^{L-\ell}\}} \sum_{\substack{C\succ B\\ \bar{t}_q\in C}} e^{-2\pi\imath\bar{\xi}_p\cdot\bar{t}_q} x(\bar{t}_q)\\
&\approx \sum_{B=\{B_j^{L-\ell}\}} \sum_{\substack{C\succ B\\ \bar{t}_q\in C}} \sum_{\xi_k\in A} e^{-2\pi\imath\left(\bar{\xi}_p-\xi_k\right)\cdot t_0^C} \mathcal{L}_k(\bar{\xi}_p) e^{-2\pi\imath\xi_k\cdot\bar{t}_q} x(\bar{t}_q)\\
&\approx \sum_{B=\{B_j^{L-\ell}\}} \sum_{\substack{C\succ B\\ \bar{t}_q\in C}} \sum_{\xi_k\in A} e^{-2\pi\imath\left(\bar{\xi}_p-\xi_k\right)\cdot t_0^C} \mathcal{L}_k(\bar{\xi}_p) \left(\sum_{\xi_s^p\in P} e^{-2\pi\imath\left(\xi_k-\xi_s^p\right)\cdot t_0^C} \mathcal{L}_s(\xi_k) e^{-2\pi\imath\xi_s^p\cdot\bar{t}_q} \right) x(\bar{t}_q)\\
&\approx \sum_{B=\{B_j^{L-\ell}\}} \sum_{\xi_k\in A} e^{-2\pi\imath\left(\bar{\xi}_p-\xi_k\right)\cdot t_0^C} \mathcal{L}_k(\bar{\xi}_p) \left(\sum_{\xi_s^p\in P} \sum_{C\succ B} e^{-2\pi\imath\left(\xi_k-\xi_s^p\right)\cdot t_0^C} \mathcal{L}_s(\xi_k) \lambda_s^{PC} \right),
\end{aligned} \tag{2.16}$$

where t_0^C denotes the center of C, the first and second approximation are due to (2.6) and the last approximation comes from the definition of λ_s^{PC}. The summations in the bracket in the last line of (2.16) defines a transference between coefficients. Hence, λ_k^{AB} defined as

$$\lambda_k^{AB} = \sum_{\xi_s^p\in P} \sum_{C\succ B} e^{-2\pi\imath\left(\xi_k-\xi_s^p\right)\cdot t_0^C} \mathcal{L}_s(\xi_k) \lambda_s^{PC}, \tag{2.17}$$

naturally satisfies (2.15).

5. *Interpolation* $(\ell = L)$. Finally, $\ell = L$, for $B = B_0^0$ and each $A = A_i^L$, we approximate the $\hat{x}(\bar{\xi}_p)$ for $\bar{\xi}_p \in A$ as

$$\begin{aligned}
\hat{x}(\bar{\xi}_p) &= \sum_{\bar{t}_q\in B} e^{-2\pi\imath\bar{\xi}_p\cdot\bar{t}_q} x(\bar{t}_q)\\
&\approx \sum_{\bar{t}_q\in B} \sum_{\xi_k\in A} e^{-2\pi\imath\left(\bar{\xi}_p-\xi_k\right)\cdot t_0} \mathcal{L}_k(\bar{\xi}_p) e^{-2\pi\imath\xi_k\cdot\bar{t}_q} x(\bar{t}_q) \approx \sum_{\xi_k\in A} e^{-2\pi\imath\left(\bar{\xi}_p-\xi_k\right)\cdot t_0} \mathcal{L}_k(\bar{\xi}_p) \lambda_k^{AB}.
\end{aligned} \tag{2.18}$$

We notice that summation kernels in (2.9) and (2.12) relies only on the relative distance of ts and, hence, it can be viewed as a convolution kernel. Similarly summation kernels in (2.17) and (2.18) relies only on the relative distance of ξs, which can be viewed as convolution kernels as well. Such an observation plays important role in the design of *Butterfly-net*.

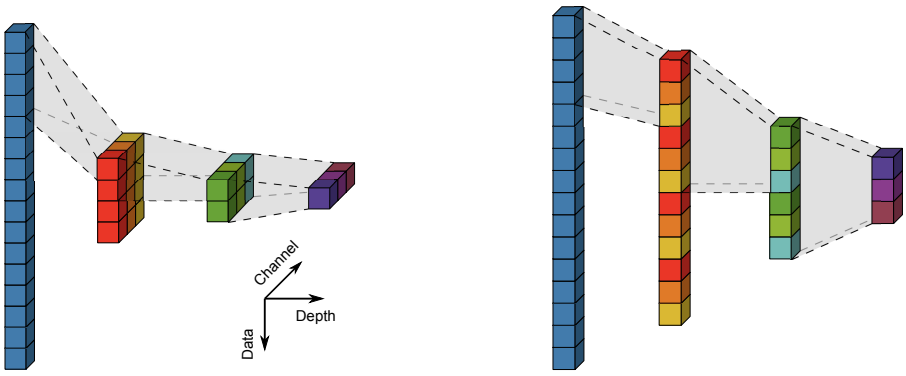
3 *Butterfly-net*

Butterfly-net is a novel structured CNN which requires far less number of parameters to accurately represent functions that can be expressed in the frequency domain. The essential building block of *Butterfly-net* is the interpolation convolutional layer, illustrated in Fig. 2 and described in Section 3.1, which by itself is also interesting as it gives another way of interpreting channel mixing. Section 3.2 assembles interpolation convolutional layers together and proposes the *Butterfly-net*. A complexity analysis is carefully derived in Section 3.3. Finally, in Section 3.4, we provide the matrix representation of the *Butterfly-net*, which significantly simplifies the analysis in Section 4.

3.1 Interpolation convolutional layer

To introduce the interpolation convolutional operation, we first introduce an equivalent formula of the usual convolutional layer by "channel unfolding", and then illustrate the layer through a hierarchical interpolation example.

Let $\{x(i,k_1)\,|\,i=0,\cdots,n-1;\ k_1=1,\cdots,c_1\}$ be a general input data with length n and c_1 channels. Assume the 1D convolutional layer maps c_1 input channels to c_2 output channels and the convolution filter is of size w. The parameters, then, can be written as



(a) 1D interpolation convolutional layers with folded channel representation

(b) 1D interpolation convolutional layers with unfolded channel representation

Figure 2: 1D interpolation convolutional layers with input size 16. The first layer is a 1D conv with filter size and stride size both being 4 and contains 3 output channels. All later layers are 1D convs with filter size and stride size being 2 and the input and output channels sizes are 3.

W_{k_2,i,k_1} for $i=0,\cdots,w-1$, $k_1=1,\cdots,c_1$, and $k_2=1,\cdots,c_2$. The output of the convolutional layer, under these notations, is written as

$$y(j,k_2)=\sum_{\substack{0\le i\le w-1\\ 1\le k_1\le c_1}} W_{k_2,i,k_1}x(i+s(j-1),k_1), \tag{3.1}$$

where $s\ge 1$ is the stride size and $j=0,\cdots,(n-w)/s$ denotes the data index of output. In many cases, it is more convenient to unfold the channel index into a vector as the input data, i.e., $x[ic_1+k_1]=x(i,k_1)$, $y[jc_2+k_2]=y(j,k_2)$, and $W[k_2,iw+k_1]=W_{k_2,i,k_1}$. Hence (3.1) can be represented as the matrix vector product,

$$y(j,:)=y[jc_2+(1:c_2)]=Wx[sc_1j+(1:wc_1)], \tag{3.2}$$

where Matlab notation is adopted. Without considering the weight sharing of bias term in the convolution layer, all channel direction can be unfolded into the data dimension and the convolution is modified as a block convolution. Such an unfolded convolutional layer will be called the interpolation convolutional layer. Interpolation convolutional layer is a way to understand the relation between channel dimension and data dimension, while, in practice, it is still implemented through regular convolutional layer.

The representation of interpolation convolutional layer is motivated by the observation that function interpolation can be naturally represented as a multi-channel convolution. In this setting, unfolding channels is more natural. The connection between function interpolation and coefficient transference in *Butterfly* algorithm, e.g., (2.9), (2.12), (2.17) and (2.18) lies in the fact that the Lagrange polynomials only depends on the relative distance of points. Hence can be realized by a convolutional kernel. Let $B_0,B_1,\cdots,B_{J-1}$ be a equal spaced partition of $[0,1)$, i.e., $B_j=[j/J,j+1/J)$, and $t_{k_1}^{B_j}$ denote the k_1-th discretization point in B_j for $k_1=1,\cdots,c_1$. We further assume that the locations of $t_{k_1}^{B_j}$ relative to B_j are the same for all j. The input data is viewed as the function $x(t)$ evaluated at the points $t_{k_1}^{B_j}$, i.e., $x(j,k_1)=x(t_{k_1}^{B_j})$. Let $t_{k_1}^{B_j}$ be the interpolation points on B_j, with the Lagrange basis polynomial given by

$$\mathcal{L}_{k_1}(s)=\prod_{\substack{p=1\\ p\neq k_1}}^{c_1}\frac{s-t_p^{B_j}}{t_{k_1}^{B_j}-t_p^{B_j}},\quad s\in B_j. \tag{3.3}$$

The interpolated function of $x(t)$ at $s_{k_2}^{B_j}$ for $k_2=1,\cdots,c_2$ is then defined as

$$\begin{aligned} x(s_{k_2}^{B_j})\approx y(j,k_2)=\sum_{k_1=1}^{c_1}\mathcal{L}_{k_1}(s_{k_2}^{B_j})x(t_{k_1}^{B_j})&=\sum_{k_1=1}^{c_1}\prod_{\substack{p=1\\ p\neq k_1}}^{c_1}\frac{s_{k_2}^{B_j}-t_p^{B_j}}{t_{k_1}^{B_j}-t_p^{B_j}}x(j,k_1)\\ &=\sum_{k_1=1}^{c_1}\prod_{\substack{p=1\\ p\neq k_1}}^{c_1}\frac{s_{k_2}^{B_0}-t_p^{B_0}}{t_{k_1}^{B_0}-t_p^{B_0}}x(j,k_1), \end{aligned} \tag{3.4}$$

where $j=0,\cdots,J-1$. The last equality in (3.4) is due to the fact that each fraction in (3.4) depends only on the relative distance and is thus independent of B_j. Therefore, we could denote $W_{k_2,i,k_1}=\prod_{p\neq k_1}{}^{(s_{k_2}^{B_0}-t_p^{B_0})}/_{(t_{k_1}^{B_0}-t_p^{B_0})}$, and thus the formula (3.4) can be interpreted as convolution (3.1) with the stride size s being the same as the filter size w, i.e., $s=w$. In this representation, the two channel indices k_1 and k_2 denote the original points and interpolation points within each domain B_j. Therefore, unfolding the channel index of both x and y leads to the natural ordering of the index of points on $[0,1)$.

For CNN with multiple convolutional layers, the unfolding of the channel index could be done recursively. Fig. 2(a) illustrates 1D interpolation convolutional layers, whereas Fig. 2(b) shows its unfolded representation. Gray zones in both figures indicate the data dependency between layers. Fig. 2(b) can also be understood as a recursive function interpolation. The domain is first divided into four subdomains and the first layer interpolates the input function within each subdomain to its three interpolation points. The layer afterwards merges two adjacent subdomains into a bigger subdomain and interpolates the function defined on the previous 6 interpolation points to the new 3 interpolation points on the merged subdomain.

If the assumption $w=s$ is removed, the convolutional layer can still be understood as an interpolation with overlapping subdomains. Similar idea is used in Simpson's rule and multi-step methods.

3.2 *Butterfly-net* architecture

This section formally introduce *Butterfly-net* architecture. We follow the exact structure of *Butterfly* algorithm here. Parallel reading of Section 2.2 and this section is recommended. For each layer, we introduce the neural network structure followed by specifying the pre-defined *Butterfly* initialization and an explanation related to the Fourier transform.

Let $x(t)$ be the input data viewed as a signal in time. Time-frequency analysis usually decomposes the signal into different modes according to frequency range, e.g., high-, medium-, low-frequency modes. Most importantly, once the signal is decomposed into different modes, they are analyzed separately and will not be mixed again. This corresponds to the non-mixing channel idea in *Butterfly-net* and the non-mixing channel has an explicit correspondence with frequency modes.

We adopt the same notations as in Section 2.2: the input vector is of length N and the output is a feature vector of length K. Let L denote the number of major layers in the *Butterfly-net*, and r denote the size mixing channels. Further L_t and $L_{\tilde{\xi}}$ denote the number of layers before and after the switch layer with $L=L_t+L_{\tilde{\xi}}$. We assume $L\leq\log N$ and $L_{\tilde{\xi}}\leq\log K$. The input tensor is denoted as $f(t,1)$ for $t=0,\cdots,N-1$. If *Butterfly* initialization is used, we construct an $L=L_t+L_{\tilde{\xi}}$ layer hierarchical partition of both domains as in Table 1. Since the input vector, the output vector, and the weights in *Butterfly* algorithm are of complex value, the connection between complex-valued operations and real-valued operations are complicated. Under ReLU activation function, the connection is detailed

in Appendix B. Throughout the *Butterfly-net*, a general tensor notation $\lambda^{(\ell)}(i,j,k)$ is used,

$$\lambda^{(\ell)}(i,j,k)=\lambda_k^{A_i^\ell B_j^{L-\ell}}, \tag{3.5}$$

corresponding to coefficients in *Butterfly* algorithm in Section 2.2. The index k denotes the mixing channel. The index i and j denote the non-mixing channel and data dimension before switch layer and denote the data dimension and non-mixing channel after switch layer. The range of index ℓ, i, and j can be found in Table 1.

Then the *Butterfly-net* architecture is described as follows under complex-valued operations.

1. *Interpolation* $(\ell=0)$. Let $m=N/2^L$ denote the filter size, which corresponds to the number of points in each B_j^L. A 1D convolution layer with filter size m, stride size m and output channel r is applied to $x(:,1)$ together with added bias term and ReLU activation. The weight tensor is denoted as $W_{k,q,1}^{(0)}$, where $k=1,\cdots,r$ and $q=1,\cdots,m$. This layer maps the input tensor x to an output tensor $\lambda^{(0)}$.

   Following the notation in (2.9), the weight tensor can be initialized as

   $$W_{k,q,1}^{(0)}\overset{\diamond}{=}e^{-2\pi\iota\xi_0\cdot\left(\bar{t}_q-t_k\right)}\mathcal{L}_k(\bar{t}_q),\quad 1\le k\le r\quad\text{and}\quad 1\le q\le m, \tag{3.6}$$

   where $\overset{\diamond}{=}$ denotes extended assign operator as defined in Appendix B.

   This step interpolates function from uniform grid points to Chebyshev points. When the frequency domain of the input signal is not symmetric around origin, this step also extracts extra phase term.

2. *Recursion* $(\ell=1,\cdots,L_t)$. The input and output tensors at layer ℓ are $\lambda^{(\ell-1)}$ and $\lambda^{(\ell)}$. For each of the non-mixing channel at previous layer, two (for $\ell\le L_{\min}$) or one (for $L_{\min}<\ell\le L_t$) 1D convolution layers with filter size 2, stride 2 and output channel r are applied together with bias term and ReLU activation. The weight tensors are denoted as and $W_{k,c,s}^{(\ell),i}$, where $k,s=1,\cdots,r$ and c corresponds to the index of child domain C of B.

   Following the notations in (2.12), the weight tensor can be initialized as

   $$W_{k,c,s}^{(\ell),i}\overset{\diamond}{=}e^{-2\pi\iota\xi_0^i\cdot(t_s^c-t_k)}\mathcal{L}_k(t_s^c), \tag{3.7}$$

   where ξ_0^i denotes the center of A_i^ℓ, t_s^c denotes the Chebyshev points in $C_c=B_c^{L-\ell+1}$ and t_k denotes the Chebyshev points in $B_0^{L-\ell}$.

   Each ξ_0^i is the center of A_i^ℓ corresponding to different frequency domain. Different frequency component in the input signal is now organized in different non-mixing channels. They will be transformed independently later which is related to the orthogonality of basis functions in different non-overlapping frequency domains.

3. *Switch* $(\ell = L_t)$. This layer is a special layer of local operations. Denote the input tensor as $\lambda^{(L_t)}(i,j,s)$ and the dense weights as $W_{k,s}^{(L_t),i,j}$ for $k,s=1,\cdots,r$ and i,j with range in Table 1. For each i,j, $W^{(L_t)}$ is a r by r dense matrix. The operation at this layer is as follows,

$$\lambda^{(L_t)}(i,j,k) = \sum_{s=1}^{r} W_{k,s}^{(L_t),i,j} \lambda^{(L_t)}(i,j,s) \tag{3.8}$$

for each pair of i,j. A bias term and ReLU layer are applied to the output tensors.

Following the notation in (2.14), the dense weight tensors can be initialized as

$$W_{k,s}^{(L_t),i,j} \triangleq e^{-2\pi\imath \xi_k^{A_i^{L_t}} \cdot t_s^{B_j^{L_\xi}}}, \tag{3.9}$$

where $\xi_k^{A_i^{L_t}}$ and $t_s^{B_j^{L_\xi}}$ are Chebyshev points in $A_i^{L_t}$ and $B_j^{L_\xi}$ respectively.

For each i,j, the Fourier operator is applied at this layer. Afterwards, interpolation is applied again in frequency domains.

4. *Recursion* $(\ell = L_t+1,\cdots,L_t+L_\xi)$. The input and output tensors at layer ℓ are $\lambda^{(\ell-1)}$ and $\lambda^{(\ell)}$. The weight tensors are denoted as and $W_{k,c,s}^{(\ell),j}$, where $k,s=1,\cdots,r$ and c corresponds to the index of child domain C of B. For each of the non-mixing channel j, one 1D convolution layer is performed as

$$\lambda^{(\ell)}(i,j,k) = \sum_{s=1}^{r} \sum_{c=0,1} W_{k,c,s}^{(\ell),j,a} \lambda^{(\ell-1)}(\lfloor i/2 \rfloor, 2j+c, s), \tag{3.10}$$

where $a = i \bmod 2$. Such a convolution is also known as transposed convolution. This transpose property will become more clear later when the matrix representation is derived.

Following the notations in (2.17), the weight tensor can be initialized as

$$W_{k,c,s}^{(\ell),j,a} \triangleq e^{-2\pi\imath \left(\xi_k^a - \xi_s\right) \cdot t_0^{2j+c}} \mathcal{L}_s(\xi_k^a), \tag{3.11}$$

where t_0^{2j+c} denotes the center of $C = B_{2j+c}^{L-\ell+1}$, ξ_s denotes the Chebyshev points in $A_0^{\ell-1}$, ξ_k^a denotes the Chebyshev points in A_a^ℓ for $a=0,1$.

This part is similar to step 3. Instead of organizing output in non-mixing frequency domains, different time component in the input signal is now organized in different non-mixing channel. This is due to the complementary property between time and frequency.

5. *Interpolation* $(\ell = L)$. Let $m = \frac{K}{2^{L_\xi + L_{\min}}}$ denote the output channel size, which corresponds to the number of points in each A_i^L. A 1D convolution layer with filter size

1, stride size 1, input channel r and output channel m is applied to $\lambda^{(L)}$ together with added bias term and ReLU activation. The weight tensor is denoted as $W^{(L)}_{p,0,k}$, where $p=1,\cdots,m$ and $k=1,\cdots,r$.

Following the notations in (2.18), the weight tensor can be initialized as

$$W^{(L)}_{p,0,k} \overset{\diamond}{=} e^{-2\pi\imath\left(\bar{\xi}_p-\bar{\zeta}_k\right)\cdot t_0}\mathcal{L}_k(\bar{\xi}_p), \quad 1\le p\le m \quad \text{and} \quad 1\le k\le r. \tag{3.12}$$

This layer generates the output tensor denoted as $\hat{x}(i,p)$, for $i=0,\cdots,2^L-1$ being the index of data and $p=1,\cdots,m$ being the index of channel. Reshaping $\hat{x}(s)=\hat{x}(i,p)$ for $s=im+p$ gives a single vector output, which is analogy of the output vector of the *Butterfly* algorithm.

6. *Task-dependent layers.* Any type of layers, e.g., dense layer, convolution layer, transpose convolution layer, etc., can be built on top of $\hat{x}$ and approximate the desired task. These layers are creatively designed by users, which are not regarded as parts of *Butterfly-net* in the following.

To further facilitate the understanding of the *Butterfly-net*, Fig. 1 demonstrates an example of the *Butterfly-net* with input vector being partitioned into 16 parts. We adopts the unfolded representation of the mixing channel as in Fig. 2(b), and the channel direction only contains non-mixing channels.

In the above description, the non-mixing channels and mixing channels are indexed different. If we combine this two indices into a single channel index, and allow all channel connections to be dense, we define another family of neural networks, namely *Inflated-Butterfly-net*. If the *Butterfly* initialization in *Butterfly-net* is applied to *Inflated-Butterfly-net* and the rest channel connections are initialized as zero, then *Inflated-Butterfly-net* is an identical operator as *Butterfly-net* with *Butterfly* initialization.

Also notice that $L_{\bar{\zeta}}$ is a tunable parameter for both *Butterfly-net* and *Inflated-Butterfly-net*. When $L_{\bar{\zeta}}=0$, all transpose convolutional layers disappear. In this case, the switch layer and interpolation layer $\ell=L$ can be combined as a entry-wise product operator, which can be implemented through a dense layer. *Inflated-Butterfly-net* with $L_{\bar{\zeta}}=0$ is then a regular CNN and *Butterfly-net* is a CNN with channel sparse structure [49]. All of our following complexity analysis and approximation analysis apply to $L_{\bar{\zeta}}=0$ with/without merging switch layer and interpolation layer $\ell=L$.

A comment on the activation function: In the above construction of *Butterfly-net*, specifically, in the extended assign operator $\overset{\diamond}{=}$ defined in Appendix B, we use the ReLU function $\mathrm{ReLU}(x)=(x)^+$ to represent the identify mapping $\mathrm{Id}(x)$ by $\mathrm{Id}(x)=\mathrm{ReLU}(x)-\mathrm{ReLU}(-x)$. Generally, we can adopt other activation function as long as the identity mapping $\mathrm{Id}(x)$ can be represented or approximated. Take sigmoid activation function as an example. The sigmoid mapping is approximately the Id mapping near origin, and thus, composing with a scaling before and another scaling afterwards, it can

approximate Id mapping as well. Revising the operations in the construction of $\stackrel{\diamond}{=}$ in Appendix B accordingly provides the *Butterfly-net* which has the same theoretical properties. With sigmoid activation function, it allows even without doubling the number of channels. Using other types of activation function with universal approximation property, the Fourier initialization scheme can be revised similarly.

3.3 Complexity analysis

One major advantage of the proposed *Butterfly-net* is the reduction of model complexity and computational complexity. We now conduct a careful count on the number of weights. Since we use complex embedding with ReLU, i.e., "$\stackrel{\diamond}{=}$", each complex multiplicative weight is actually implemented by a 4×4 matrix and each bias is implemented as a vector of length 4. Hence $4r$ is the actual number of channels. We conduct the counting layer by layer based on the network description and Table 1. On the interpolation ($\ell=0$), there is only one convolutional kernel, which has $16r\cdot\frac{N}{2^L}$ filter weights and $4r$ bias weights. On the recursive ($\ell=1,\cdots,L_t$), there are $\min(2^{\ell-1},2^{L_{\min}})$ non-mixing input channels, $\min(2^{\ell},2^{L_{\min}})$ non-mixing output channels, and the filter size is 2. The number of both input and output mixing channels are $4r$. Hence there are $\min(2^{\ell},2^{L_{\min}})\cdot(4r)^2\cdot 2$ filter weights and $\min(2^{\ell},2^{L_{\min}})\cdot 4r$ bias weights. On the switch layer ($\ell=L_t$), there are $\min(2^{L_t},2^{L_{\min}})$ non-mixing channels and 2^{L_ξ} data. Since the connection is locally fully connected layer, the overall number of multiplicative weights is $\min(2^{L_t},2^{L_{\min}})\cdot 2^{L_\xi}(4r)^2$ with additional $\min(2^{L_t},2^{L_{\min}})\cdot 2^{L_\xi}4r$ bias weights. On the recursive layer ($\ell=L_t+1,\cdots,L_t+L_\xi$), it is a transpose of the previous recursive layer $2L_t-\ell$. The numbers of filter weights and bias weights are then $2^{L-\ell}\cdot(4r)^2\cdot 2$ and $2^{L-\ell}\cdot 4r$. The last interpolation ($\ell=L$) is similar as the interpolation ($\ell=0$). The numbers of filter weights and bias weights are $4\max(1,\frac{K}{2^L})(4r)$ and $4\max(1,\frac{K}{2^L})$. In summary, the overall network complexity for *Butterfly-net* is

$$\begin{aligned}&\frac{N}{2^L}16r+\sum_{\ell=1}^{L_t}\min\left(2^{\ell},\frac{K}{2^{L_\xi}}\right)(4r)^2\cdot 2+\min(2^L,K)(4r)^2+\sum_{\ell=L_t+1}^{L}2^{L-\ell}(4r)^2\cdot 2+\max\left(1,\frac{K}{2^L}\right)16r\\&\quad+4r+\sum_{\ell=1}^{L_t}\min\left(2^{\ell},\frac{K}{2^{L_\xi}}\right)4r+\min(2^L,K)4r+\sum_{\ell=L_t+1}^{L}2^{L-\ell}4r+\max\left(1,\frac{K}{2^L}\right)4r\\&=\mathcal{O}\left(\frac{Nr}{2^L}+LKr^2\right),\end{aligned}\tag{3.13}$$

where the first row is the number of multiplicative/filter weights and the second row is the number of bias weights.

Remark 3.1. We further derive a more precise upper bound for the number of weights in *Butterfly-net* with an absolute constant under $L=\log N$. Denote the expression in (3.13) as n_b. We have

$$n_b\le 40r+L_t\frac{K}{2^{L_\xi}}(32r^2+4r)+K(16r^2+4r)+2^{L_\xi}(32r^2+4r)<90LKr^2,\tag{3.14}$$

Table 2: Leading order network complexity for *Butterfly-net* and *Inflated-Butterfly-net* under two scenarios, $K\sim N\sim 2^L$ and $K=\mathcal{O}(1), N\sim 2^L$. In both cases, we assume $K\leq N$ and $L_\xi\leq \log K$, i.e. $L-\log K\leq L_t\leq L$. The asymptotic regime is $N\to\infty$, and r denotes the number of interpolation points (size of mixing channels) which does not change with N.

	$K\sim N\sim 2^L$		$K=\mathcal{O}(1)$, $N\sim 2^L$	
	BNet	Inflated *BNet*	*BNet*	Inflated *BNet*
Interpolation ($\ell=0$)	r	r	r	r
Recursion ($\ell\leq L_t$)	$2^\ell r^2$	$2^{2\ell}r^2$	$\frac{K}{2^{L_\xi}}r^2$	$\frac{K^2}{2^{2L_\xi}}r^2$
Switch ($\ell=L_t$)	$2^L r^2$	$2^L r^2$	Kr^2	Kr^2
Recursion ($\ell\geq L_t+1$)	$2^{L-\ell}r^2$	$2^{2(L-\ell)}r^2$	$2^{L-\ell}r^2$	$2^{2(L-\ell)}r^2$
Interpolation ($\ell=L$)	r	r	r	r
Overall	$2^L r^2$	$2^{2L_t}r^2$	$L_t\frac{K}{2^{L_\xi}}r^2+Kr^2$	$L_t\frac{K^2}{2^{2L_\xi}}r^2+Kr^2$
$L_\xi=0$	Nr^2	N^2r^2	$Kr^2\log N$	$K^2r^2\log N$
$L_\xi=\frac{1}{2}\log K$	Nr^2	Nr^2	$\sqrt{K}r^2\log N+Kr^2$	$Kr^2\log N$

where the last inequality adopts the assumption on r as in Theorem 2.1.

The number of parameters in the inflated *Butterfly-net*, i.e., all channels are mixed, can be derived in a similar way and the overall complexity is

$$\begin{aligned}
&\frac{N}{2^L}16r+\sum_{\ell=1}^{L_t}\min\Big(2^\ell,\frac{K}{2^{L_\xi}}\Big)^2(4r)^2\cdot 2+\min(2^L,K)(4r)^2\\
&\quad+\sum_{\ell=L_t+1}^{L}2^{2L-2\ell}(4r)^2\cdot 2+\max\Big(1,\frac{K}{2^L}\Big)16r+4r+\sum_{\ell=1}^{L_t}\min\Big(2^\ell,\frac{K}{2^{L_\xi}}\Big)4r\\
&\quad+\min(2^L,K)4r+\sum_{\ell=L_t+1}^{L}2^{L-\ell}4r+\max\Big(1,\frac{K}{2^L}\Big)4r\\
=&\mathcal{O}\Big(\frac{Nr}{2^L}+L\frac{K^2}{2^{2L_\xi}}r^2+2^{2L_\xi}r^2\Big).
\end{aligned}\tag{3.15}$$

In Table 2, we summarize the layer-wise complexity together with the overall complexity for two special scenarios, $K\sim N\sim 2^L$ and $K=\mathcal{O}(1), N\sim 2^L$, for both *Butterfly-net* and *Inflated-Butterfly-net*, where we use the notation $A\sim B$ to denote that $\frac{A}{B}\to c$ which is an absolute strictly positive constant in the limit of $N\to\infty$. Recall that the number of hierarchical layer L satisfies that $K\leq 2^L$.

When *Butterfly-net* is extended to d-dimension, we should partition both the time and the frequency domains into 2^d subdomains at each layer and use r^d Chebyshev points for interpolations. All above detailed derivations for complexity analysis can be updated accordingly. The overall complexity remain the same as that in Table 2 with r being replaced by r^d.

In addition to the network complexity, the overall computational cost for a evaluation of the *Butterfly-net* and the inflated *Butterfly-net* are $\mathcal{O}(N\log N)$ and $\mathcal{O}(N\log N+K^2)$.

3.4 Matrix representation of *Butterfly-net*

This section aims to demonstrate the matrix representation of *Butterfly-net*, which is similar to *Butterfly* factorization [32]. The matrix representation further explain the sparse connectivity of channels and more importantly facilitates the proof in the analysis of approximation power in Section 4.

We first show the matrix representation of the interpolation convolutional layer and the matrix representation of the *Butterfly-net* simply stacks the interpolation convolutional layer together with a switch layer. Fig. 3(a) represents (3.2) when both g and f are vectorized. If we permute the row ordering of the matrix, we would result blocks of convolution matrix with the number of blocks being the size of the output channels. Fig. 3(b) assumes that $s=w$ and the matrix is further simplified to be a block diagonal matrix. According to the figures, we note that when $s=w$, the transpose of the matrix is a representation of a transposed convolutional layer with W replaced by W^T.

Since the convolutional layer with mixing channel can already be represented as Fig. 3, *Butterfly-net* stack interpolation convolutional layer together representing non-mixing channel, which is equivalent to stack the matrix row-wise. We would explain the matrix representation for each step of the *Butterfly-net*.

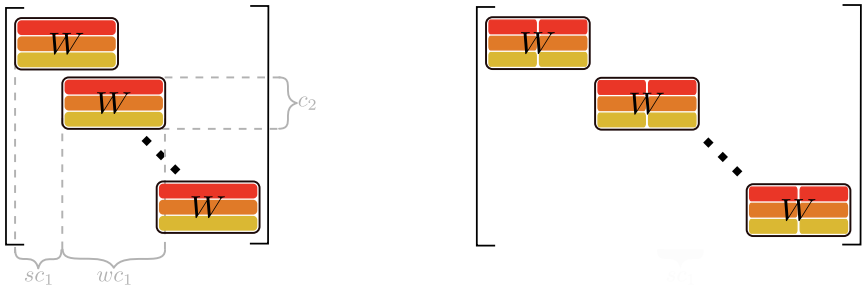
1. *Interpolation* $(\ell=0)$. The matrix representation is Fig. 3(b) with 2^L diagonal blocks and each block is $W^{(0)}_{:,:,1}$ which is of size $r\times m$. The resulting matrix is denoted as V.

2. *Recursion* $(\ell=1,\cdots,L_t)$. The weight tensor $W^{(\ell),i}_{k,c,s}$ can be reshaped as a matrix with row indexed by k and column indexed by c and s, which is denoted as $W^{(\ell),i}$. For $\ell\leq L_{\min}$ the convolutional operator of mixing and non-mixing channels can be viewed as the following matrices respectively,

$$H^{(\ell)}_{\lfloor i/2\rfloor}=\begin{pmatrix} W^{(\ell),2\lfloor i/2\rfloor} & & \\ & \ddots & \\ & & W^{(\ell),2\lfloor i/2\rfloor} \\ \hline W^{(\ell),2\lfloor i/2\rfloor+1} & & \\ & \ddots & \\ & & W^{(\ell),2\lfloor i/2\rfloor+1} \end{pmatrix}, \text{ and } H^{(\ell)}=\begin{pmatrix} H^{(\ell)}_0 & & \\ & \ddots & \\ & & H^{(\ell)}_{2^{\ell-1}-1} \end{pmatrix}. \quad (3.16)$$

For $L_{\min}<\ell\leq L_t$ the convolutional operator can be viewed as the following matrices respectively,

$$H^{(\ell)}_i=\begin{pmatrix} W^{(\ell),i} & & \\ & \ddots & \\ & & W^{(\ell),i} \end{pmatrix}, \quad \text{and} \quad H^{(\ell)}=\begin{pmatrix} H^{(\ell)}_0 & & \\ & \ddots & \\ & & H^{(\ell)}_{2^{L_{\min}}-1} \end{pmatrix}. \quad (3.17)$$



(a) Matrix of 1D interp. conv. layer (b) Matrix of 1D interp. conv. layer with $s=w$

Figure 3: Matrix representation of 1D interpolation convolutional layers.

3. *Switch* $(\ell=L_t)$. The matrix representation of switch layer is

$$M=\begin{pmatrix} M_{0,0} & \cdots & M_{0,2^{L_t}-1} \\ \vdots & \ddots & \vdots \\ M_{2^{L_\xi}-1,0} & \cdots & M_{2^{L_\xi}-1,2^{L_t}-1} \end{pmatrix}. \tag{3.18}$$

Each $M_{i,j}$ is again an $2^{L_\xi}\times 2^{L_t}$ block matrix with block size $r\times r$, where all blocks are zero except that the (j,i) block is $W_{:,:}^{(L_t),i,j}$.

4. *Recursion* $(\ell=L_t+1,\cdots,L_t+L_\xi)$. The weight tensor $W_{k,c,s}^{(\ell),j,a}$ can be reshaped as a matrix with row indexed by k,a and column indexed by s, which is denoted as $W_c^{(\ell),j}$. Then the convolutional operator of mixing and non-mixing channels can be viewed as the following matrices respectively,

$$G_j^{(\ell)}=\left(\begin{array}{ccc|ccc} W_0^{(\ell),j} & & & W_1^{(\ell),j} & & \\ & \ddots & & & \ddots & \\ & & W_0^{(\ell),j} & & & W_1^{(\ell),j} \end{array}\right), \text{ and } G^{(\ell)}=\begin{pmatrix} G_0^{(\ell)} & & \\ & \ddots & \\ & & G_{2^{L-\ell}-1}^{(\ell)} \end{pmatrix}. \tag{3.19}$$

5. *Interpolation* $(\ell=L)$. The matrix representation is Fig. 3(b) with 2^L diagonal blocks and each block is $W_{:,0,:}^{(L)}$ which is of size $\min(1,\frac{K}{2^L})\times r$. The resulting matrix is denoted as U.

Based on the matrix representation of each layer of the *Butterfly-net*, we could write down the overall matrix representation together with bias terms and ReLU activation layer as

$$y=\mathcal{B}(x)=U\tilde{\sigma}\left[G^{(L)}\tilde{\sigma}\left[\cdots G^{(L_t+1)}\tilde{\sigma}\left[M\tilde{\sigma}\left[H^{(L_t)}\tilde{\sigma}\left[\cdots H^{(1)}\tilde{\sigma}[Vx]\right]\right]\right]\right]\right], \tag{3.20}$$

where $U,G^{(\ell)},M,H^{(\ell)},V$ are defined as above and $\tilde{\sigma}[\cdot]$ denotes the operation of properly adding the bias and applying the ReLU activation.

4 Analysis of approximation power

The main result of function approximation using *Butterfly-net* is given in Section 4.1, where the main Theorem, Theorem 4.2, relies on the Fourier kernel approximation result Corollary 4.1. The latter is established in Section 4.2, and the proof is in Section 4.3.

4.1 Approximation of network function

In this section, we first focus on approximating single-valued function $f(\vec{x})$, where $\vec{x}\in\mathbb{R}^N$, N is an integer. After showing the main result Theorem 4.2, we discuss on the extension to other prediction functions, particularly dense predictions in U-net.

Let F_N be the N-by-N unitary discrete Fourier matrix. Any vector $\vec{x}$ and its discrete Fourier transform $\hat{x}=F_N\vec{x}$ obey $\|\vec{x}\|_2=\|\hat{x}\|_2$. By normalizing argument, we assume that $\vec{x}$ lies in $\mathcal{X}$, which is a subset of the 2-norm unit ball in $\mathbb{R}^N$. As a result, $\hat{x}\in\hat{\mathcal{X}}$ is also contained in the 2-norm unit ball in $\mathbb{C}^N$. Suppose that $\vec{x}$ is statistically distributed according to $dP(x)$ on $\mathcal{X}$. We denote the 2-norm ball of radius r in $\mathbb{C}^m$ as $\hat{B}_r^m$, and that in $\mathbb{R}^m$ as B_r^m.

We introduce the following assumption on f and $\mathcal{X}$ which, as shown in Theorem 4.2, can be more efficiently represented using *Butterfly-net*.

Assumption 4.1. Consider approximating f under the p-norm of $L^p(\mathcal{X},dP)$, $p\in[1,\infty]$. There exists constants $C_2>0$, $s>0$, an interval $I_K=[K_0,K_0+K)$ for $K\leq N$, and a function g_K which maps from $\hat{B}_{1.1}^K$ to $\mathbb{R}$, such that the following three conditions hold.

(i) Let $\hat{x}|_{I_K}$ denote retrieving entries of the vector $\hat{x}$ with index in I_k,

$$\|f(\vec{x})-g_K(\hat{x}|_{I_K})\|_{L^p(\mathcal{X},dP),\vec{x}\in\mathcal{X}}<0.1,$$

and note $\hat{x}|_{I_K}\in\hat{B}_1^K$ since $\hat{x}\in\hat{B}_1^N$.

(ii) g_K is C_2-Lipschitz on $\hat{B}_{1.1}^K$ with respect to the 2-norm, i.e.,

$$|g_K(z)-g_K(z')|\leq C_2\|z-z'\|_2,\quad \forall z,\, z'\in\hat{B}_{1.1}^K.$$

Note that if we view g_K as a real-input function taking $2K$ inputs, it also has Lipschitz constant C_2 (with respect to the 2-norm in real space). Here C_2 is uniformly for all K.

(iii) For any $0<\varepsilon<1$, there is a multi-layer fully-connected network with number of parameters

$$n_{fc}\leq c(s,K)\varepsilon^{-K/s}\Big(\log\frac{1}{\varepsilon}+1\Big), \tag{4.1}$$

which gives a network mapping $\phi_K:\hat{B}_{1.1}^K\to\mathbb{R}$ s.t. $\|\phi_K(z)-g_K(z)\|_{\infty,z\in\hat{B}_{1.1}^K}<\varepsilon$, where $c(s,K)$ is a constant that depends on s and K.

Remark 4.1. In Assumption 4.1 (i), the constant 0.1 is technical and can be any other constant less than one. As will be shown in Theorem 4.2, the approximation result is most useful when the residual norm in Assumption 4.1 (i) can be made small, say $\varepsilon \ll 1$, and then this ε will be used as the target ε in both the *Butterfly-net* approximation and the fully-connected network approximation. Note that if f is band-limited to begin with, the residual can be made zero. For frequency decaying function, the residual can be made arbitrarily small by increasing K. Detailed in the examples below.

Example 4.1. Consider the energy functional of the 1D Laplace operator with periodic boundary condition

$$f(\vec{x}) = E(\vec{x}) = \sum_{1 \le k \le N/2} \frac{2}{k^2} |\hat{x}_k|^2, \quad \vec{x} \in \mathcal{X} = B_1^N,$$

and dP is some distribution of $\vec{x}$ on $\mathcal{X}$. For any $K > 0$, setting $I_K = [1, K]$, and

$$g_K(\hat{x}|_{I_K}) = \sum_{1 \le |k| \le K} \frac{2}{k^2} |\hat{x}_k|^2.$$

(i) We can verify that

$$\|f(\vec{x}) - g_K(\hat{x}|_{I_K})\|_{L^p(\mathcal{X}, dP), \vec{x} \in \mathcal{X}} < 2 \cdot K^{-3/2},$$

which decays as K increases. This means that the residual can be made arbitrarily close to 0 if K can be chosen sufficiently large.

(ii) Due to that g_K is a quadratic form, and $\|\hat{x}|_{I_K}\|_2 \le 1$, Assumption 4.1 (ii) is satisfied with $C_2 = 2.2$.

(iii) Viewing g_K as a function taking $2K$ real input and defined on $B_{1.1}^{2K}$, it is again a quadratic form. Then by setting $s = 2$, or even higher positive number, the neural network approximation theory in [50] provides the uniform approximation needed in Assumption 4.1 (iii) by a network whose model complexity is bounded by (4.1), where the constant $c(s, K)$ depends on the space dimension $2K$, the derivative order s, and the Sobolev norm of the function g_K in $\mathcal{W}^s(B_{1.1}^{2K})$.

Example 4.2. Unlike the first example which makes use of the form of f, this example mainly shifts assumptions on the data set. Suppose the data domain $\mathcal{X}$ consists of band-limited data points with window length K,

$$\mathcal{X} = \{\vec{x} \in B_1^N \mid \hat{x}_k = 0 \text{ for } k \notin I_K\}.$$

Then for a regular function f defined on $B_{1.1}^N$, e.g., C_2-Lipschitz and s-smooth, let $\mathcal{K}^{-1}$ be the discrete inverse Fourier transform from frequency window I_K to $\mathcal{X} \subset \mathbb{R}^N$, and $g_K(z) = f(\mathcal{K}^{-1} z)$ for any $z \in \hat{B}_{1.1}^K$. Hence the difference in Assumption 4.1 (i) is zero and

the regularity assumptions on g_K can be inherited from that on f. Thus (ii) and (iii) in Assumption 4.1 are also satisfied.

In applications, if the input data vectors $\vec{x}$ are discretizations of regular continuous signals, then they naturally have spectrum decay property and can be approximated by band-limited data points. Hence the Assumption 4.1 also applies.

Remark 4.2. Our theory leaves the abstract approximation bound in Assumption 4.1 (iii) and focuses on the approximation analysis of *Butterfly-net*, of which the key result is Corollary 4.1. Generally, as long as a "nice" component g_K, which is band-limited on a Fourier window of length K and has sufficient regularity, can be separated out from f up to a small residual, the universal approximation theory of neural network gives approximation of g_K depending on its regularity, and this fully-connected network is used on top of the *Butterfly-net*.

Theorem 4.2. *Assume that the function f and $\mathcal{X}$ satisfy Assumption* 4.1*, and notations are the same as therein. Then for any ε satisfying*

$$\|f(\vec{x})-g_K(\hat{x}|_{I_K})\|_{L^p(\mathcal{X},dP),\vec{x}\in\mathcal{X}}\leq\varepsilon<0.1,$$

there exists a family of Butterfly-nets with $L_\xi=0,\cdots,\log K$, whose numbers of parameters are all bounded by

$$n_b\leq CK\log N\left(\log\frac{1}{\varepsilon}+\log N\right)^2,\quad C \text{ being an absolute constant,}$$

and a fully connect network ϕ_K with number of parameters

$$n_{fc}\leq c(s,K)\varepsilon^{-K/s}\left(\log\frac{1}{\varepsilon}+1\right),$$

such that using any member of the family of Butterfly-net, denoting by $\mathcal{B}$, gives

$$\|f(\vec{x})-\phi_K(\mathcal{B}(\vec{x}))\|_{L^p(\mathcal{X},dP),\vec{x}\in\mathcal{X}}\leq\varepsilon(2+C_2).$$

Remark 4.3. Theorem 4.2 holds for a family of *Butterfly-net*s with different L_ξs, which shows no difference in view of this approximation analysis. However, different L_ξs lead to *Butterfly-net* architectures with different number of parameters. *Butterfly-net* with smaller L_ξ has more parameters (up to a logarithmic factor), hence achieves better post-training accuracy, which is verified in Section 5.2. *Butterfly-net* with $L_\xi=0$ has the simplest architecture consisting of only convolutional layers and fully-connect layers, which is detailed in [49].

Remark 4.4. Since *Butterfly-net* can be embedded in *Inflated-Butterfly-net*, Theorem 4.2 also holds for a family of *Inflated-Butterfly-net* with $L_\xi=0,\cdots,\log K$, whose numbers of parameters are all bounded by

$$n_b\leq C\left(\frac{K^2}{2^{2L_\xi}}\log N+2^{2L_\xi}\right)\left(\log\frac{1}{\varepsilon}+\log N\right)^2. \tag{4.2}$$

Notice the number of parameters first increases and then decrease as $L_{\tilde{\xi}}$ varying from 0 to $\log K$. The lightest *Inflated-Butterfly-net* is achieved when $L_{\tilde{\xi}}=\frac{\log K}{2}$. As shown in [49], *Inflated-Butterfly-net* with $L_{\tilde{\xi}}=0$ is a regular CNN. Hence, Theorem 4.2 gives an approximation result for regular CNNs if *Butterfly-net* is replaced by *Inflated-Butterfly-net* and the parameter number upper bound on n_b is replaced by (4.2).

Remark 4.5. Theorem 4.2 can be extended to approximate functions of $\vec{x}$ with $\vec{x}$ being a function discretized on higher dimensional grids. As been discussed around Theorem 2.1, the key low-rank approximation can be generalized to higher dimensions if high dimensional Chebyshev interpolations are applied. Following Section 3.2 and [33], the architecture of multidimensional Butterfly factorization can be implemented using high dimensional CNNs with/without sparse channel connections. And high dimensional CNNs can be initialized to approximate high dimensional Fourier kernels. The approximation bound, then, is similar to that in Theorem 4.3 with a modified prefactor depending on the dimensionality. Given a high dimensional version of Theorem 4.3 and a high dimensional extension of Assumption 4.1, Theorem 4.2 can be extended to high dimensional settings and viewed as an approximation upper bound for high dimensional CNNs.

Proof. Under Assumption 4.1, for any $\vec{x}\in\mathcal{X}$,

$$|f(\vec{x})-\phi_K(\mathcal{B}(\vec{x}))|\leq|f(\vec{x})-g_K(\hat{x}|_{I_K})|+|g_K(\hat{x}|_{I_K})-g_K(\mathcal{B}(\vec{x}))|+|g_K(\mathcal{B}(\vec{x}))-\phi_K(\mathcal{B}(\vec{x}))|, \tag{4.3}$$

and by Corollary 4.1, there exists a family of *Butterfly-net*s satisfying the requirement and for any $\mathcal{B}$ belonging to the family,

$$\|\mathcal{B}(\vec{x})-\hat{x}|_{I_K}\|_2<\varepsilon<0.1,\quad\forall\vec{x}\in\mathcal{X}, \tag{4.4}$$

due to that $\mathcal{X}\subset\hat{B}_1^N$. Note that the $\mathcal{K}$ operator in Corollary 4.1 is defined for frequency window, c.f. beginning of Section 4.2, and thus $\hat{x}|_{I_K}$ equals $\mathcal{K}\vec{x}$. Combined with that $\hat{x}|_{I_K}\in\hat{B}_1^K$, both $\hat{x}|_{I_K}$ and $\mathcal{B}(\vec{x})$ must lie in $\hat{B}_{1.1}^K$ on which g_K and ϕ_K are defined. We now bound the three terms in (4.3) respectively.

The 2nd term: By that $\hat{x}|_{I_K},\mathcal{B}(\vec{x})\in\hat{B}_{1.1}^K$, Assumption 4.1 (ii) gives that

$$|g_K(\hat{x}|_{I_K})-g_K(\mathcal{B}(\vec{x}))|\leq C_2\|\hat{x}|_{I_K}-\mathcal{B}(\vec{x})\|_2<C_2\varepsilon,$$

where the second inequality uses (4.4). This pointwise upper bound leads to the p-norm bound of this term to be $C_2\varepsilon$ for all p.

The 3rd term: By the uniform approximation of g_K on $\hat{B}_{1.1}^K$ guaranteed by Assumption 4.1 (iii), the fully-connected network ϕ_K satisfying the requirement exists, and pointwisely,

$$|g_K(\mathcal{B}(\vec{x}))-\phi_K(\mathcal{B}(\vec{x}))|<\varepsilon.$$

This proves that the p-norm bound of this term is smaller than ε for all p.

The 1st term: the p-norm is at most ε by assumption.

Putting together proves the p-norm upper bound in the claim. □

Remark 4.6. When the p-norm residual in Assumption 4.1 (i) can be made zero, which includes the case of band-limited f and Example 4.2, the ε in the Theorem 4.2 can be chosen arbitrarily close to zero and the final approximation bound can be made $\varepsilon(1+C_2)$.

Note that the above results demonstrate the power of *Butterfly-net* when $K<N$. Assume the function f and the frequency truncated g_K has similar regularity level, which is typically true as shown in both Example 4.1 and Example 4.2. Then a fully-connected network approximating the original function $f:\mathcal{X}\to\mathbb{R}$, requires a total number of parameter

$$n_{fc}\leq c(s,N)\varepsilon^{-N/s}\left(\log\frac{1}{\varepsilon}+1\right).$$

As a comparison, the overall number of parameter for *Butterfly-net* with a fully-connected network is

$$n_b+n_{fc}\leq CK\log N\left(\log\frac{1}{\varepsilon}+\log N\right)^2+c(s,K)\varepsilon^{-K/s}\left(\log\frac{1}{\varepsilon}+1\right),$$

where the second term which involves $\varepsilon^{-K/s}$ dominates the model complexity. The improvement from $\varepsilon^{-N/s}$ to $\varepsilon^{-K/s}$ is due to the feature extraction of truncated Fourier coefficients which are suitable for the class of functions as described in Assumption 4.1. In other words, while N being the ambient dimensionality of the input data, K upper-bounds the intrinsic complexity (effective dimension) of the regression problem f on the data $\mathcal{X}$.

Discussion on the dense prediction networks. The method of analyzing x-to-1 network functions in this section can extend to the analysis of x-to-x function mappings, known as dense prediction in deep learning literature. Typical deep convolutional networks for x-to-x prediction include the U-net architecture [46], which consists of a module of multiple convolutional layers, a bottleneck module with dense connections, and another module of multiple conv-t layers, namely transpose convolutional layers. The corresponding network architecture using *Butterfly-net* would be $\mathcal{B}$-(fc layers)-$\mathcal{B}^T$ where $\mathcal{B}$ denotes a module of *Butterfly-net* layers, and $\mathcal{B}^T$ is again a *Butterfly-net* module due to the symmetric role of time of frequency in the model. Extending the approximation theory, such U-net which replaces traditional convolutional layers by *Butterfly-net* layers can provably approximate operators in the form of $\mathcal{F}^{-1}\circ\sigma\circ\mathcal{F}$, where $\mathcal{F}$ is Fourier transform operator, and σ is some non-linear operator (not necessarily entry-wise). The covered family of operators involve low/high-pass filtering, de-convolution, among others, in signal processing.

Another example of dense prediction mapping is the general Laplace operator (with variable coefficient) in physics which can be represented as $\sum_i \mathcal{A}_i\mathcal{K}\mathcal{D}_i\mathcal{F}$, where i sums over a small number of terms corresponding to a low-rank decomposition of the amplitude function, $\mathcal{A}_i$s and $\mathcal{D}_i$s are diagonal matrices, and $\mathcal{K}$ is a smooth Fourier integral operator (FIO) [10, 15]. As a heads-up, a parallel reading of the proof of Theorem 4.3

and [10,31] reveals that similar theorem can be provided for smooth FIOs, i.e., $e^{-2\pi\iota\Phi(\xi,t)}$ with smooth $\Phi(\xi,t)$ satisfying homogeneity condition of degree 1. Such an extension of Theorem 4.3 will give error control of *Butterfly-net* approximation of the operator $\mathcal{K}$, and $\mathcal{A}_i$s, $\mathcal{D}_i$s can be represented by fully-connected networks, possibly coordinate-separated and shallow. The approximation to smooth FIOs will enable the usages of *Butterfly-net* to represent a large class of elliptic operators.

4.2 Approximation of the Fourier kernel

In this section, we analyze the approximation power of the *Butterfly-net* on discrete Fourier kernel, whose matrix entry is defined as $\mathcal{K}_{ij}=e^{-2\pi\iota\xi_i\cdot t_j}$ where t_j and ξ_i are uniformly distributed on $[0,1)$ and $[k_0,k_0+K)$ ($K\leq N$) respectively. The analysis result shows that though *Butterfly-net* by construction has very low complexity as the number of parameters is on the order of the input/output data size, it exhibits full approximation power in terms of function representations.

Theorem 4.3. *Let N denote the size of the input and K denote the size of the output in the Butterfly-net. L and r are two parameters such that $\pi eK\leq r2^{\min(L,\log K)}$. L is the depth of the Butterfly-net and r is the size of mixing channels. There exists a parametrized Butterfly-net, $\mathcal{B}(\cdot)$, approximating the discrete Fourier transform such that for any bounded input vector $\vec{x}$, the error of the output of the Butterfly-net satisfies that for any $p\in[1,\infty]$,*

$$\|\mathcal{K}\vec{x}-\mathcal{B}(\vec{x})\|_p\leq m^{\frac{1}{p}}r^{L_t(1-\frac{1}{p})+L_\xi\frac{1}{p}+1}(2\Lambda_r)^{L+3}\frac{K(\pi e)^r}{r^{r-1}}\|\vec{x}\|_p, \tag{4.5}$$

where $m=\min(1,K/2^L)$ and $\Lambda_r=\frac{2}{\pi}\ln r+1$ is the Lebesgue constant.

If $L\leq\log K$, then the error also satisfies

$$\|\mathcal{K}\vec{x}-\mathcal{B}(\vec{x})\|_p\leq C_{r,K}\left(\frac{\Lambda_r}{2^{r-2}}\right)^L r^{L_t(1-\frac{1}{p})+L_\xi\frac{1}{p}}\|\vec{x}\|_p, \tag{4.6}$$

where $C_{r,K}=(2\Lambda_r)^3\frac{(\pi eK)^r}{(2r)^{r-1}}$ is a constant depending on r and K, and independent of L.

The proof of Theorem 4.3 is constructive. We first fill the *Butterfly-net* with a specific set of parameters (*Butterfly* initialization) based on the complementary low-rank property of the discrete Fourier kernel (see Theorem 2.1). Using the matrix representation of *Butterfly-net*, 1-norm and ∞-norm of each matrix can be bounded. Combined with the low-rank approximation error, we derive the 1-norm and ∞-norm upper bound for *Butterfly-net*. Applying Riese-Thorin interpolation theorem, we reach to the conclusion of Theorem 4.3 for general norm index p. Section 4.3 provides the detailed proof of the theorem.

Previously, in the context of fast algorithms, Kunis and Melzer [29] analyzed the approximations of a simplified *Butterfly* scheme and Demanet et al. [16] analyzed general *Butterfly* scheme under different error measures on the input and output. While as a side

product of our proof, we also obtain the error estimate of the matrix approximation of the general *Butterfly* schemes in terms of matrix norms.

For a problem with fixed input and output size, we can tune two parameters r and L to reach desired accuracy. As r increases, which corresponds the increase of mixing channel size in each layer, the approximation error decays mainly as r^{-r}. Interestingly, when L increases, which corresponds to increase the depth of the *Butterfly-net*, the error bound decays exponentially in L.

Combining Theorem 4.3 and the parameter number estimation (3.14), we derive the network complexity analysis for *Butterfly-net* under a given approximation accuracy as follows.

Corollary 4.1. *Let N denote the size of the input and K denote the size of the output in the Butterfly-net. For any $0<\varepsilon<1$, there exists a family of Butterfly-nets with $L_\xi=0,1,\cdots,\log K$, whose numbers of parameters are all bounded by*

$$n_b \leq CK\log N\left(\log\frac{1}{\varepsilon}+\log N\right)^2, \tag{4.7}$$

where $C<(2\pi e)^2\cdot 90$ is an absolute constant, and for any $\mathcal{B}$ denoting a Butterfly-net in the family,

$$\|\mathcal{K}\vec{x}-\mathcal{B}(\vec{x})\|_2<\varepsilon\|\vec{x}\|_2, \tag{4.8}$$

for any input vector $\vec{x}$.

Proof. Comparing the statements in Corollary 4.1 and Theorem 4.3, we aim to find a r such that

$$T=\sqrt{\min(1,\frac{K}{2^L})}\sqrt{r}^{L+2}\left(2+\frac{4}{\pi}\ln r\right)^{L+3}\frac{K(\pi e)^r}{r^{r-1}}<\varepsilon. \tag{4.9}$$

In this proof, we assume that $L=\log N$. Since we focus on the approximation error under $p=2$ norm, the prefactor T is then independent of L_ξ and r becomes the only tunable parameter in *Butterfly-net*. The following proof holds for all $0\leq L_\xi\leq\log K$.

When $r\geq\alpha(\log N+\log\frac{1}{\varepsilon})$, we have

$$T\leq\left(\frac{2\sqrt{r}\left(2+\frac{4}{\pi}\ln r\right)}{\left(\frac{r}{\pi e}\right)^\alpha}\right)^L\frac{r^2\left(2+\frac{4}{\pi}\ln r\right)^3}{\left(\frac{r}{\pi e}\right)^{\alpha-1}}\frac{1}{\left(\frac{r}{\pi e}\right)^{\log\frac{1}{\varepsilon}}}. \tag{4.10}$$

For both the first and second term in (4.10) the denominator grows faster in r than the numerator. When $\alpha=2\pi e$, both the first and second terms are smaller than one for all $r\geq\alpha$ [‡]. Then we have when $r\geq 2\pi e(\log N+\log\frac{1}{\varepsilon})$,

$$T<\frac{1}{\left(\frac{r}{\pi e}\right)^{\log\frac{1}{\varepsilon}}}<\frac{1}{2^{\log\frac{1}{\varepsilon}}}=\varepsilon. \tag{4.11}$$

[‡]Both inequalities are numerically validated as well.

Based on the network complexity counting in (3.14), there exists an absolute constant $C_0=90$, and r such that the number of parameters in *Butterfly-net* is

$$n_b \leq C_0 K \log N r^2. \tag{4.12}$$

Substituting the conditions for r into (4.12), we conclude that there exists a *Butterfly-net* with the number of parameters bounded by

$$n_b \leq CK\log N\left(\log\frac{1}{\varepsilon}+\log N\right)^2, \quad C\leq(2\pi e)^2 C_0, \tag{4.13}$$

such that $T<\varepsilon$, which proves the theorem. □

4.3 Proof of Theorem 4.3

This section first provides a few lemmas and their proof bounding each sparse matrix in (3.20). And then Theorem 4.3 is proved in detail.

Lemma 4.1. *Let $\{z_i\}_{i=1}^r$ be r Chebyshev points and $\mathcal{L}_k(x)$ be the Lagrange polynomial of order r. For any r, the Lebesgue constant Λ_r is bounded as*

$$\Lambda_r = \max_{-1\leq x\leq 1}\sum_{i=1}^{r}|\mathcal{L}_i(x)| \leq \frac{2}{\pi}\ln r+1.$$

Lemma 4.1 is a standard result of Chebyshev interpolation and the proof can be found in [45].

Corollary 4.2. *Let $\{z_i\}_{i=1}^r$ be r Chebyshev points and $\mathcal{L}_k(x)$ be the Lagrange polynomial of order r. For any r and $i\leq r$,*

$$\max_{-1\leq x\leq 1}|\mathcal{L}_i(x)| \leq \frac{2}{\pi}\ln r+1.$$

Corollary 4.2 is an immediate result of Lemma 4.1.

Lemma 4.2. *Let U be the block diagonal matrix defined at interpolation layer ($\ell=L$), then*

$$\|U\|_1 \leq m\left(\frac{2}{\pi}\ln r+1\right) \quad \textit{and} \quad \|U\|_\infty \leq \frac{2}{\pi}\ln r+1,$$

where r is the number of Chebyshev points and $m=K/2^L$.

Proof. U is a block diagonal matrix with block U_i. U_i are the same $m\times r$ with entry $e^{-2\pi\imath(\bar{\xi}_p-\xi_k)\cdot t_0}\mathcal{L}_k(\bar{\xi}_p)$ for $\bar{\xi}_p$ and ξ_k being uniform and Chebyshev points in A_i^L. By the definition of matrix 1-norm, we have

$$\|U\|_1=\|U_1\|_1 \leq \max_{\xi_k\in A_1^L}\sum_{\bar{\xi}_p\in A_1^L}\left|e^{-2\pi\imath(\bar{\xi}_p-\xi_k)\cdot t_0}\mathcal{L}_k(\bar{\xi}_p)\right| \leq m\left(\frac{2}{\pi}\ln r+1\right). \tag{4.14}$$

By the definition of matrix ∞-norm, we have

$$\|U\|_\infty = \|U_1\|_\infty \le \max_{\bar{\xi}_p \in A_1^L} \sum_{\xi_k \in A_1^L} \left| e^{-2\pi\imath(\bar{\xi}_p - \xi_k)\cdot t_0} \mathcal{L}_k(\bar{\xi}_p) \right| \le \frac{2}{\pi}\ln r + 1. \tag{4.15}$$

This completes the proof. □

Lemma 4.3. *Let M be the block matrix defined at switch layer, then*

$$\|M\|_1 \le r \quad \textit{and} \quad \|M\|_\infty \le r,$$

where r is the number of Chebyshev points.

Proof. Based on the structure of M and the definition of matrix 1-norm, we have

$$\|M\|_1 \le \max_j \max_i \|W_{j,i}\|_1 = \max_{j,i} \max_{t_k^{B_j^{L_\xi}}} \sum_{\xi_{k'}^{A_i^{L_t}}} \left| e^{-2\pi\imath \xi_{k'}^{A_i^{L_t}} \cdot t_k^{B_j^{L_\xi}}} \right| = r. \tag{4.16}$$

Based on the structure of M and the definition of matrix ∞-norm, we have

$$\|M\|_\infty \le \max_j \max_i \|W_{j,i}\|_\infty = \max_{j,i} \max_{t_k^{A_i^{L_t}}} \sum_{t_k^{B_j^{L_\xi}}} \left| e^{-2\pi\imath \xi_{k'}^{A_i^{L_t}} \cdot t_k^{B_j^{L_\xi}}} \right| = r. \tag{4.17}$$

This completes the proof. □

Lemma 4.4. *Let $H^{(\ell)}$ be the block diagonal matrix defined at recursion layer $\ell = 1, \cdots, L_t$, then*

$$\left\| H^{(\ell)} \right\|_1 \le 2\left(\frac{2}{\pi}\ln r + 1 \right),$$

where r is the number of Chebyshev points.

Proof. The building block of $H^{(\ell)}$ is $W_i^{(\ell)}$, whose 1-norm is bounded as

$$\left\| W_i^{(\ell)} \right\|_1 \le \max_{t \in B_1^{L-\ell}} \sum_{t_k^{B_1^{L-\ell}}} \left| e^{-2\pi\imath \xi_0^{A_i^\ell} \cdot (t - t_k^{B_1^{L-\ell}})} \mathcal{L}_k(t) \right| \le \frac{2}{\pi}\ln r + 1. \tag{4.18}$$

For two ranges of ℓ, i.e., $\ell \le L_{\min}$ and $L_{\min} < \ell \le L_t$, we have

$$\left\| H^\ell \right\|_1 = \max_i \left\| H_i^{(\ell)} \right\|_1 \le 2 \left\| W_i^{(\ell)} \right\|_1 \le 2\left(\frac{2}{\pi}\ln r + 1 \right). \tag{4.19}$$

This completes the proof. □

Lemma 4.5. *Let $G^{(\ell)}$ be the block diagonal matrix defined at recursion layer $\ell=L_t+1,\cdots,L_t+L_{\tilde{\zeta}}$, then*

$$\left\|G^{(\ell)}\right\|_1 \leq 2r\left(\frac{2}{\pi}\ln r+1\right),$$

where r is the number of Chebyshev points.

Proof. Based on the structure of $G^{(\ell)}$ and the definition of matrix 1-norm, we have

$$\begin{aligned}\left\|G^{\ell}\right\|_1 &= \max_j \left\|G_j^{(\ell)}\right\|_1 \leq \max_j \left\|W_j^{(\ell)}\right\|_1 \\ &\leq \max_j \max_k \left(\sum_{\tilde{\zeta}_{k'}^{A_1^{\ell-1}}} \left|\mathcal{L}_k(\tilde{\zeta}_{k'}^{A_1^{\ell-1}})\right| + \sum_{\tilde{\zeta}_{k'}^{A_2^{\ell-1}}} \left|\mathcal{L}_k(\tilde{\zeta}_{k'}^{A_2^{\ell-1}})\right|\right) \leq 2r\left(\frac{2}{\pi}\ln r+1\right). \end{aligned} \tag{4.20}$$

This completes the proof. □

Lemma 4.6. *Let $H^{(\ell)}$ be the block diagonal matrix defined at recursion layer $\ell=1,\cdots,L_t$, then*

$$\left\|H^{(\ell)}\right\|_\infty \leq 2r\left(\frac{2}{\pi}\ln r+1\right),$$

where r is the number of Chebyshev points.

Lemma 4.7. *Let $G^{(\ell)}$ be the block diagonal matrix defined at recursion layer $\ell=L_t+1,\cdots,L_t+L_{\tilde{\zeta}}$, then*

$$\left\|G^{(\ell)}\right\|_\infty \leq 2\left(\frac{2}{\pi}\ln r+1\right),$$

where r is the number of Chebyshev points.

The proofs of Lemma 4.6 and Lemma 4.7 follow that of Lemma 4.5 and Lemma 4.4 respectively.

Proof of Theorem 4.3. Since all bias weights are initialized as zero, and extensive assign is applied together with ReLU activation, *Butterfly-net* with *Butterfly* initialization is equivalent to multiplying the matrices in matrix representation together, i.e.,

$$\mathcal{B}(x)=UG^{(L)}\cdot G^{(L_t+1)}MH^{(L_t)}\cdot H^{(1)}Vx. \tag{4.21}$$

We then write the exact matrix product with error matrices $E^{(\ell)}$,

$$\mathcal{K}=E^{(L+1)}+U\left[E^{(L)}+G^{(L)}\left[\cdots+G^{(L_t+1)}\left[E^{(S)}+M\left[E^{(L_t)}+H^{(L_t)}\left[\cdots+H^{(1)}\left[E^{(0)}+V\right]\right]\right]\right]\right]\right]. \tag{4.22}$$

Then, following properties of matrix p-norm, we have

$$\begin{aligned}\varepsilon_p &= \left\| \mathcal{K} - UG^{(L)}\cdots G^{(L_t+1)}MH^{(L_t)}\cdots H^{(1)}V \right\|_p \\ &\le \left\| E^{(L+1)} \right\|_p + \left\| UE^{(L)} \right\|_p + \cdots + \left\| UG^{(L)}\cdots G^{(L_t+1)}E^{(S)} \right\|_p + \cdots \\ &\quad + \left\| UG^{(L)}\cdots G^{(L_t+1)}MH^{(L_t)}\cdots H^{(1)}E^{(0)} \right\|_p \\ &\le \left\| E^{(L+1)} \right\|_p + \|U\|_p \left\| E^{(L)} \right\|_p + \cdots + \|U\|_p \left(\prod_{\ell=L_t+1}^{L} \left\| G^{(\ell)} \right\|_p \right) \left\| E^{(S)} \right\|_p + \cdots \\ &\quad + \|U\|_p \left(\prod_{\ell=L_1+1}^{L} \left\| G^{(\ell)} \right\|_p \right) \|M\|_p \left(\prod_{\ell=1}^{L_t} \left\| H^{(\ell)} \right\|_p \right) \left\| E^{(0)} \right\|_p, \end{aligned} \tag{4.23}$$

for $1 \le p \le \infty$.

When matrix 1-norm is used, we adopt Lemma 4.2, Lemma 4.3, Lemma 4.4, and Lemma 4.5 to bound ε_1 as

$$\varepsilon_1 \le \delta_1 \left(1 + m\Lambda_r \sum_{\ell=0}^{L_\xi} (2r\Lambda_r)^\ell + mr\Lambda_r (2r\Lambda_r)^{L_\xi} \sum_{\ell=0}^{L_t} (2\Lambda_r)^\ell \right) \le m r^{L_\xi+1} (2\Lambda_r)^{L+2} \delta_1, \tag{4.24}$$

where the second inequality adopt the fact that $r>1$ and $\Lambda_r = \frac{2}{\pi}\ln r + 1() > 1$, δ_1 is a uniform upper bound for $\|E^{(\ell)}\|_1$. Theorem 2.1 provides an upper bound for each entry of $E^{(\ell)}$. Recalling the vector length ($\sharp\{i\}\cdot\sharp\{j\}\cdot r$) and product of domain lengths from Table 1, the uniform upper bound for the 1-norm follows,

$$\delta_1 = \max_\ell \left\| E_1^{(\ell)} \right\|_1 \le \pi e K (1+\Lambda_r) \left(\frac{\pi e K}{r 2^{L_\xi + L_{\min}+1}} \right)^{r-1}, \tag{4.25}$$

where the assumption $\pi e K \le r 2^{\min(\log K, L)}$ is applied.

When matrix ∞-norm is used, we adopt Lemma 4.2, Lemma 4.3, Lemma 4.6, and Lemma 4.7 to bound ε_∞ as

$$\varepsilon_\infty \le \delta_\infty \left(1 + \Lambda_r \sum_{\ell=0}^{L_\xi} (2\Lambda_r)^\ell + r\Lambda_r (2\Lambda_r)^{L_\xi} \sum_{\ell=0}^{L_t} (2r\Lambda_r)^\ell \right) \le r^{L_t+1} (2\Lambda_r)^{L+2} \delta_\infty, \tag{4.26}$$

where δ_∞ is a uniform upper bound for $\left\| E^{(\ell)} \right\|_\infty$ and obeys the same upper bound as δ_1.

Applying Riesz-Thorin interpolation theorem together with (4.24) and (4.26), we obtain the error bound under the matrix p-norm

$$\varepsilon_p \le m^{\frac{1}{p}} r^{L_t(1-\frac{1}{p})+L_\xi \frac{1}{p}+1} (2\Lambda_r)^{L+3} \frac{K(\pi e)^r}{r^{r-1}}, \tag{4.27}$$

for any $1 \le p \le \infty$.

If we further assume $L \leq \log K$, then $2^{L_{\tilde{\xi}}+L_{\min}} = 2^L$ and the matrix p-norm error bound can be rederived in terms of L, i.e.,

$$\varepsilon_p \leq C_{r,K} \left(\frac{\Lambda_r}{2^{r-2}} \right)^L r^{L_t(1-\frac{1}{p})+L_{\tilde{\xi}}\frac{1}{p}}, \tag{4.28}$$

where $C_{r,K} = (2\Lambda_r)^3 \frac{(\pi e K)^r}{(2r)^{r-1}}$ is a constant depending on r and K, and independent of L.

For any bounded vector $\vec{f}$, Theorem 4.3 is the direct result of (4.27) and (4.28). □

5 Numerical results

We present four numerical experiments to demonstrate the approximation power with or without training for *Butterfly-net* and *Inflated-Butterfly-net*. The first numerical experiment shows that the approximation error without training of an *Butterfly* initialized *Butterfly-net* decays exponentially as the increases of the network depth L, which verifies Theorem 4.3. Then, through the second experiment, we show that the training of *Butterfly-net* and *Inflated-Butterfly-net* further refine the approximation error. The approximation error after training depends on both the properties of the function and the dataset. In the third experiment, we apply both *Butterfly-net*s and *Inflated-Butterfly-net*s to testing datasets with different distribution comparing to the training datasets and compare the transfer learning capabilities. In the last experiment, *Butterfly-net* and *Inflated-Butterfly-net* with additional task layers, one with single fully connect layer and another with square layer, are tested and compared in the approximation of the energy functionals of Poisson's equation. All algorithms are implemented using Tensorflow 2.1.0 [1] and can be found on the authors' homepages.

5.1 Approximation power without training

The first numerical experiment in this section aims to verify the exponential decay of the approximation error of the *Butterfly-net* as the depth L increases. We construct a *Butterfly-net* to approximate the discrete Fourier kernel with fixed number of Chebyshev points, r=8. The *Butterfly-net* is filled with *Butterfly* initialization weights. The input vector in this example is of size $N = 1024$ and various output vector sizes are tested. The output vector represents integer frequency of the input function in the frequency domain $[0,K)$. The approximation error of the *Butterfly-net* is measured against the dense discrete Fourier kernel matrix and relative matrix p-norm error is reported, $\epsilon_p = \|\mathcal{K}-\mathcal{B}\|_p / \|\mathcal{K}\|_p$, where $\mathcal{B}$ denotes *Butterfly-net*.

Table 3 shows for both choices of K, the relative approximation errors measured in 1-norm, 2-norm, and ∞-norm decay exponentially as L increases and stay constant for different $L_{\tilde{\xi}}$. The decay factors for different K remain similar, while the prefactor is larger for large K. All of these observations agree with the error bound in Theorem 4.3.

Table 3: Approximation accuracy of the Fourier kernel by the *Butterfly-net*. The input vector size is $N=1024$, the number of Chebyshev points is $r=8$, and frequency domain is $[0,K)$.

$K=64$					$K=256$				
L	L_ξ	ϵ_1	ϵ_2	ϵ_∞	L	L_ξ	ϵ_1	ϵ_2	ϵ_∞
	1	2.06e−1	2.46e−1	2.56e−1		1	2.52e−1	3.40e−1	2.82e−1
4	2	2.02e−1	2.60e−1	2.66e−1	6	2	2.51e−1	3.45e−1	2.89e−1
	3	1.90e−1	2.89e−1	2.72e−1		3	2.46e−1	3.60e−1	2.95e−1
	1	1.79e−3	2.56e−3	2.31e−3		1	2.03e−3	3.40e−3	2.44e−3
5	2	1.69e−3	2.32e−3	1.84e−3	7	2	1.97e−3	3.33e−3	2.01e−3
	3	1.61e−3	2.16e−3	1.94e−3		3	1.91e−3	3.15e−3	2.11e−3
	1	9.21e−6	1.30e−5	1.94e−5		1	1.15e−5	2.01e−5	2.00e−5
6	2	8.90e−6	1.33e−5	1.76e−5	8	2	1.13e−5	2.04e−5	1.82e−5
	3	8.65e−6	1.49e−5	1.70e−5		3	1.10e−5	2.07e−5	1.77e−5

5.2 Approximation power after training

The second numerical experiment in this section aims to demonstrate the approximation power after training of *Butterfly-net*.

Dataset setup. Both the training and testing datasets are generated as follows. We first generate an array of N random complex number in the frequency domain with each number sampled uniformly from $[-1,1)$. Then, we multiply the array by a Gaussian function centered at G_{center} and width G_{width} (the standard deviation of the Gaussian function). The input data $\vec{x}$ is then the real part of the inverse discrete Fourier transform of the array. Given a frequency window $[k_0,k_0+K)$, the output data is the discrete Fourier transform of $\vec{x}$ restricted to the frequency window.

We perform numerical results on four groups of datasets as shown in Table 4 with their short names. Notice that DFT-Lfreq and DFT-Hfreq have the same input data distribution and the corresponding widths are very large. Hence the input data of these two groups are close to white noise. However, DFTSmooth-Lfreq and DFTSmooth-Hfreq have input data generated with Gaussian centered at 0 and 256 with small width 10. The input data is then close to band limited signal around the given frequency window, and in other words, data in dimension N actually lie on a low-dimensional subspace of dimension about $6G_{\text{width}}=60$. In Appendix C, we include one instance for each datasets in Table 4.

Training and evaluation setup. All *Butterfly-net*s and *Inflated-Butterfly-net*s with different initializations and frequency windows are trained under the infinity data setting, i.e., training data is randomly generated on the fly. The input data length is $N=1024$, the batch size is 256, the maximum number of iteration is 50,000, and ADAM optimizer is used with an exponentially decay learning rate. The initial learning rate is 10^{-3} and 10^{-4}

Table 4: Datasets setups and their short names.

Short Name	G_{center}	G_{width}	Freq Window
DFT-Lfreq	0	500	[0,128)
DFT-Hfreq	0	500	[256,384)
DFTSmooth-Lfreq	0	10	[0,128)
DFTSmooth-Hfreq	256	10	[256,384)

Table 5: Numerical results of *Butterfly-net*s (*BNet*) and *Inflated-Butterfly-net*s (*IBNet*) on DFT-Lfreq, DFT-Hfreq, DFTSmooth-Lfreq, and DFTSmooth-Hfreq datasets. Prefix initialization refers to the *Butterfly* initialization defined in Section 3. Pre-training and post-training relative errors are reported for each dataset. Both *Butterfly-net* and *Inflated-Butterfly-net* use 16 mixing channels ($r=4$) and $L=8$ layers.

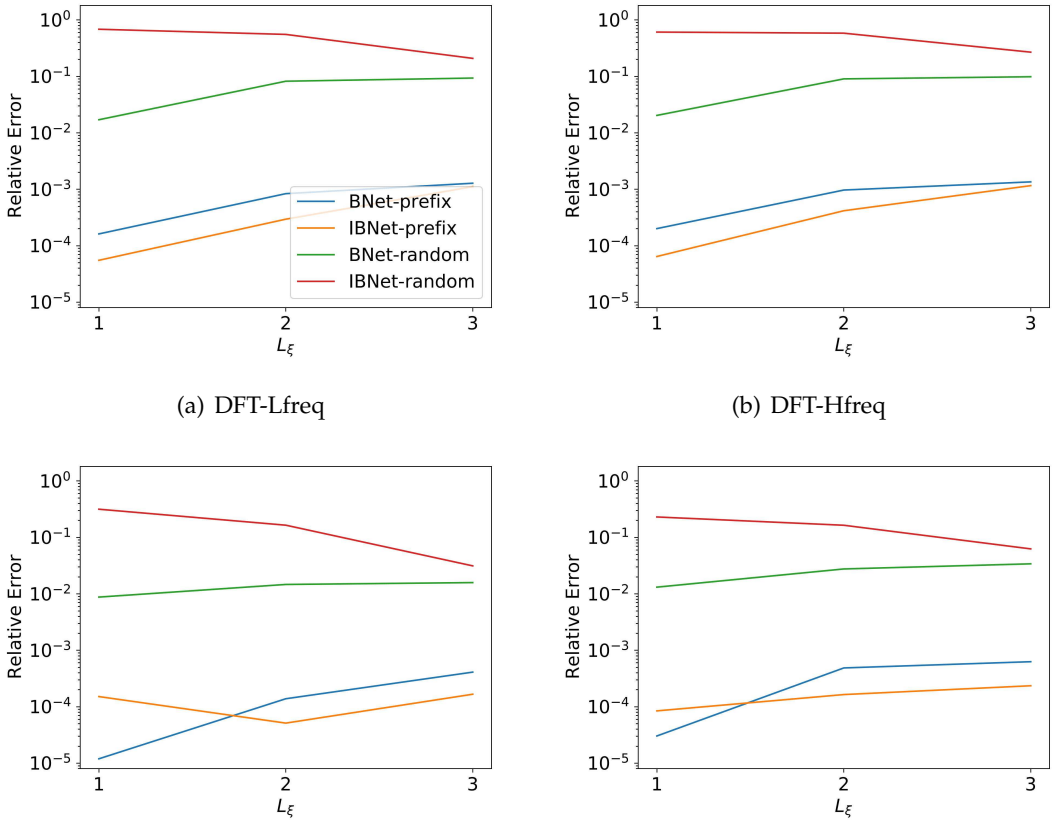
$L_{\tilde{\xi}}$	Neural Network	Initial	Num Paras	DFT-Lfreq		DFT-Hfreq		DFTSmooth-Lfreq		DFTSmooth-Hfreq	
				Pre Train	Post Train	Pre Train	Post Train	Pre Train	Post Train	Pre Train	Post Train
1	*BNet*	prefix	136304	1.9e−2	1.6e−4	1.9e−2	2.0e−4	1.9e−2	1.2e−5	2.0e−2	3.0e−5
		random	136304	1.0e−0	1.7e−2	1.0e−0	2.0e−2	1.0e−0	8.8e−3	1.0e−0	1.3e−2
	IBNet	prefix	3533936	1.9e−2	5.5e−5	1.9e−2	6.5e−5	1.9e−2	1.5e−4	1.9e−2	8.5e−5
		random	3533936	1.0e−0	6.9e−1	1.0e−0	6.1e−1	1.0e−0	3.2e−1	1.0e−0	2.3e−1
2	*BNet*	prefix	87728	1.9e−2	8.4e−4	2.0e−2	9.7e−4	2.0e−2	1.4e−4	2.0e−2	4.9e−4
		random	87728	1.0e−0	8.2e−2	1.0e−0	9.1e−2	1.0e−0	1.5e−2	1.0e−0	2.8e−2
	IBNet	prefix	915120	1.9e−2	3.0e−4	1.9e−2	4.2e−4	2.0e−2	5.1e−5	2.0e−2	1.6e−4
		random	915120	1.0e−0	5.6e−1	1.0e−0	5.8e−1	1.0e−0	1.6e−1	1.0e−0	1.6e−1
3	*BNet*	prefix	66608	2.2e−2	1.3e−3	2.2e−2	1.4e−3	2.2e−2	4.1e−4	2.2e−2	6.3e−4
		random	66608	1.0e−0	9.4e−2	1.0e−0	9.9e−2	1.0e−0	1.6e−2	1.0e−0	3.4e−2
	IBNet	prefix	275504	2.2e−2	1.1e−3	2.2e−2	1.2e−3	2.2e−2	1.7e−4	2.2e−2	2.4e−4
		random	275504	1.0e−0	2.1e−1	1.0e−0	2.7e−1	1.0e−0	3.1e−2	1.0e−0	6.3e−2

for random initialized neural networks and *Butterfly* initialized ones respectively. The decay steps and the decay rate are 100 and 0.985. The maximum number of iteration is sufficient for the convergence of relative errors in all settings (see Appendix D for examples of convergence behaviors). The loss function is defined as

$$\ell(\{\vec{x}_i,\vec{y}_i\}) = \sum_i \|\mathcal{N}(\vec{x}_i) - \vec{y}_i\|_2^2, \tag{5.1}$$

where $\vec{y}_i$ is the output data and $\mathcal{N}$ denotes a neural network. Relative errors are reported for comparison. In the following, the pre-training relative error is evaluated on the first batch and the post-training relative error is evaluated on a testing data of size 1000. Default values are used for other unspecified hyper parameters.

Results. Table 5 reports the result on all four datasets. Fig. 4 further illustrates the post-training relative errors against $L_{\tilde{\xi}}$, where $L_{\tilde{\xi}}$ indicates the position of the switch layer. We summarize the experimental observations as follows.



(a) DFT-Lfreq (b) DFT-Hfreq

(c) DFTSmooth-Lfreq (d) DFTSmooth-Hfreq

Figure 4: Post-training relative error against $L_{\tilde{\xi}}$ for all datasets. $L_{\tilde{\xi}}$ indicates the position of the switch layer. All four figures share the same legend as (a).

(1) (Weight initialization, random v.s. prefix) In all cases, training from *Butterfly* initialization significantly outperforms training from random initialization by two to three digits. In almost all cases, the post-training relative errors of the randomly initialized networks are not even as accurate as the pre-training relative error of the *Butterfly* initialized counterparts.

(2) (Channel sparsity, *Butterfly-net* v.s. *Inflated-Butterfly-net*) Given a fixed $L_{\tilde{\xi}}$, *Butterfly-net* has much less parameters than *Inflated-Butterfly-net*. While, their post-training relative errors stay at a similar level. For these datasets, channel sparsification reduces the number of parameters without loss much post-training accuracy.

(3) (Influence of datasets) Given the same input data and different output frequency windows, as in DFT-Lfreq and DFT-Hfreq, the post-training relative errors of high frequency output are slightly less accurate than their low-frequency counterparts.

The impact of output frequency window on the training performance is limited. Given the same output frequency window and different input data, i.e., DFT-Lfreq v.s. DFTSmooth-Lfreq, and DFT-Hfreq v.s. DFTSmooth-Hfreq, the post-training relative errors on datasets with smoother input data are half digit to one digit more accurate than that on datasets with less smooth input data.

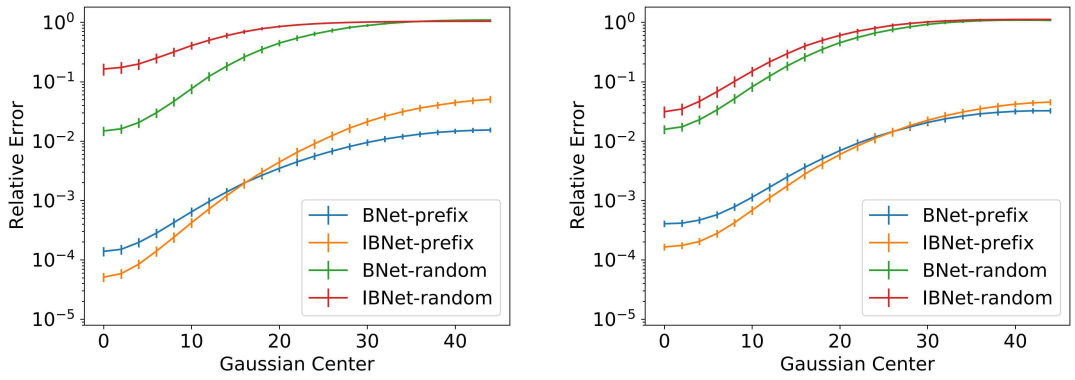
(4) (Position of switch layer, $L_{\tilde{\xi}}$) The number of parameters increases as $L_{\tilde{\xi}}$ decreases for both *Butterfly-net* and *Inflated-Butterfly-net*. In general, the post-training relative error increases as $L_{\tilde{\xi}}$ increases for all *Butterfly-net* results. However, for *Inflated-Butterfly-net* with random initialization, the post-training relative error decreases as $L_{\tilde{\xi}}$ increases on all datasets.

We further give a discussion on the above results. The comparison of random initialization and *Butterfly* initialization indicates that the trainings reach two different local minima. The relative error associated with *Butterfly* initialization is much smaller. Numerically, we also found that if we set the learning rate for *Butterfly* initialized training to be large then the training loss first increases and then converges to a number worse than the results given in the Table 5. Hence, we conjecture that the energy landscape of this task is nasty and the *Butterfly* initialization lies in a narrow but deep well. Training from *Butterfly* initialization with small learning rate achieves the local minima within this narrow but deep well. Regarding the influence of datasets, the intrinsic dimensionality of the input data plays a crucial role in training. If no guidance of feature selection is provided on an intrinsically high-dimensional data, e.g., *Inflated-Butterfly-net* with $L_{\tilde{\xi}}=1$ and random initialization applied to DFT-Lfreq, the training fails to learn useful information. Adding more guidance, either refining network structure (increasing $L_{\tilde{\xi}}$, adding channel sparsification) or providing *Butterfly* initialization, helps training and achieves lower post-training relative error. However, if the intrinsic dimension of the input data is lowered, the training learns more information and achieves lower relative error. Providing extra guidance further helps training. For the position of switch layer, we can further decrease $L_{\tilde{\xi}}$ to zero and end up with the neural network proposed in [49]. We emphasize that when $L_{\tilde{\xi}}=0$, the *Inflated-Butterfly-net* is indeed a regular CNN. More numerical results and connections between *Inflated-Butterfly-net* and regular CNN refer to [49].

5.3 Comparison to CNN and transfer learning

This numerical experiment is to compare the *Butterfly-net* and *Inflated-Butterfly-net* on their transfer learning capability. We adopt the neural networks trained on DFTSmooth-Lfreq in previous section here. A sequence of testing datasets are generated in the same way as in previous section with $G_{\text{center}}=0,2,\cdots,44$ and $G_{\text{width}}=10$. Each testing dataset contains 1000 samples.

Fig. 5 shows the means and standard deviations of the transferred testing relative errors of neural networks with $L_{\tilde{\xi}}=2$ and $L_{\tilde{\xi}}=3$. For every transferred testing, the standard deviation is orders of magnitude smaller than the corresponding mean. Hence the



(a) $L_{\xi}=2$ (b) $L_{\xi}=3$

Figure 5: Mean and standard deviation for testing relative errors of *Butterfly-net* and *Inflated-Butterfly-net* on testing datasets with various $G_{\text{center}}=0,2,\cdots,44$ and $G_{\text{width}}=10$. All networks are trained on DFTSmooth-Lfreq dataset ($G_{\text{center}}=0$ and $G_{\text{width}}=10$).

variation of the testing relative error for each setting is relatively small. Regarding the transfer learning capabilities, all randomly initialized networks quick loss accuracy as the testing dataset shifted away from the training dataset. While, *Butterfly* initialized neural networks still preserve reasonable level of accuracy. Further, comparing *Butterfly-net* and *Inflated-Butterfly-net* both with *Butterfly* initialization, *Inflated-Butterfly-net* achieves lower testing relative error when the testing dataset has significant overlap with the training dataset. However, when the overlap reduces, the testing relative error of *Inflated-Butterfly-net* increase faster than that of *Butterfly-net*. Hence we conclude that *Butterfly-net* has better generalizability than that of *Inflated-Butterfly-net*. Further, comparing the case of $L_{\xi}=2$ and $L_{\xi}=3$, we notice that the crossover of two testing relative error curves comes later in $L_{\xi}=3$. This is because that the *Inflated-Butterfly-net* with $L_{\xi}=3$ has much less number of parameters than that of *Inflated-Butterfly-net* with $L_{\xi}=2$, hence is less adapted to training dataset after training.

5.4 General function approximation

The last numerical example aims to construct an approximation of the energy functional of 1D Poisson's equation and another analog functional for high frequency input data, both of which correspond directly to the approximation power in Section 4.1 in representing general functions.

Functional setup. For a Poisson's equation $\Delta u(t)=x(t)$ with periodic boundary condition, the energy functional of Poisson's equation is defined as the negative inner product of u and x, which can also be approximated by a quadratic form of the leading low-

frequency Fourier components,

$$\mathcal{E}_1(x) = -\langle x,u\rangle \approx \sum_{k\in(-K,K)} \frac{1}{|k|^2}|\widehat{x}_k|^2 = \sum_{k\in[1,K)} \frac{2}{|k|^2}|\widehat{x}_k|^2, \tag{5.2}$$

where $\widehat{x}_k$ is the Fourier component of x at frequency k and the last equality comes from the assumption of real input x. If the input function x is a band limited function within the frequency window, $(-K,K)$, then equality is achieved in (5.2). The other analogy functional for input data x is defined as

$$\mathcal{E}_2(x) = \sum_{k\in(K_0,K_0+K)} \frac{2}{|k-K_0|^2}|\widehat{x}_k|^2. \tag{5.3}$$

Throughout this section, K_0 is 256 and K is 128. The input data is generated in the same way as that in Section 5.2. The centers of Gaussian are $G_{\text{center}} = 0$ and $G_{\text{center}} = 256$ for $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. The widths of the Gaussian for both functionals are $G_{\text{width}} = 30$.

Neural network setup. Task layers are attached to *Butterfly-net* and *Inflated-Butterfly-net*. We have two different task layers, namely square-sum-layer and dense-dense-layer. The square-sum-layer first squares all output of the previous layers in *Butterfly-net* or *Inflated-Butterfly-net*, then multiplies each squared value by a weight, and finally sums them together. This is equivalent to square the output and then connect a single dense layer with one output unit. The square-sum-layer is able to exactly represent both functionals, (5.2) and (5.3) if the weights of the dense layer is properly initialized. The dense-dense-layer attaches a dense layer with 256 output units with both bias and ReLU activation function enabled. Then another dense layer with one output unit is attached afterwards. Both functionals can only be approximated by the dense-dense-layer.

Training and evaluation setup. All *Butterfly-net*s and *Inflated-Butterfly-net*s with different initializations are trained under the infinity data setting. The input data length is $N = 1024$, the batch size is 256, the maximum number of iteration is 50,000, and ADAM optimizer is used with an exponentially decay learning rate. The initial learning rate is 10^{-4} for *Butterfly* initialized networks with square-sum-layer, and is 10^{-3} for all other settings. The decay steps and the decay rate are 100 and 0.985 respectively. Again, the maximum number of iteration is sufficient for the convergence of relative errors in all settings. The loss function is defined as

$$\ell(\{\vec{x}_i\}) = \sum_i |\mathcal{N}(\vec{x}_i) - \mathcal{E}(\vec{x}_i)|^2, \tag{5.4}$$

for $\mathcal{E}$ being either $\mathcal{E}_1$ or $\mathcal{E}_2$. Relative errors are reported for comparison. The testing data is of size 1000. Default values are used for other unspecified hyper parameters.

Result. Table 6 show the results for functional (5.2) and (5.3). The comparison of the number of parameters for *Butterfly-net*/*Inflated-Butterfly-net* is the same as that in Section 5.2, while the number of parameters in dense-dense-layer is much larger than that

Table 6: Numerical results of *Butterfly-net*s and *Inflated-Butterfly-net*s with various task layers for functional (5.2) and (5.3). Pre-training and post-training relative errors are reported for each functional. All neural networks use 16 mixing channels and $L=8$ layers.

Task Layer	L_ξ	Neural Network	Initial	Num Paras		Functional $\mathcal{E}_1$		Functional $\mathcal{E}_2$	
				BNet/*IBNet*	Task	Pre Train	Post Train	Pre Train	Post Train
Square-sum-layer	1	*BNet*	prefix	136304	256	1.80e−2	2.35e−5	1.84e−2	3.01e−5
			random	136304	256	1.00e−0	7.62e−3	1.00e−0	1.85e−2
		IBNet	prefix	3533936	256	1.69e−2	1.78e−4	1.80e−2	1.92e−4
			random	3533936	256	1.00e−0	5.80e−3	1.00e−0	9.48e−3
	2	*BNet*	prefix	87728	256	1.79e−2	4.36e−5	1.72e−2	5.84e−5
			random	87728	256	1.00e−0	9.62e−3	1.00e−0	1.74e−2
		IBNet	prefix	915120	256	1.78e−2	6.33e−5	1.66e−2	7.30e−5
			random	915120	256	1.00e−0	8.71e−3	1.00e−0	1.02e−2
	3	*BNet*	prefix	66608	256	1.50e−2	9.51e−5	1.55e−2	2.27e−4
			random	66608	256	1.00e−0	1.46e−2	1.00e−0	4.04e−2
		IBNet	prefix	275504	256	1.59e−2	5.22e−5	1.69e−2	1.17e−4
			random	275504	256	1.00e−0	1.15e−2	1.00e−0	2.38e−2
Dense-dense-layer	1	*BNet*	prefix	136304	66048	1.00e−0	7.44e−3	9.99e−1	9.30e−3
			random	136304	66048	1.00e−0	1.27e−2	1.00e−0	2.26e−2
		IBNet	prefix	3533936	66048	9.99e−1	4.11e−3	1.00e−0	5.84e−3
			random	3533936	66048	1.00e−0	7.67e−3	1.00e−0	1.44e−2
	2	*BNet*	prefix	87728	66048	9.97e−1	8.37e−3	1.00e−0	1.18e−2
			random	87728	66048	1.00e−0	2.30e−2	1.00e−0	2.19e−2
		IBNet	prefix	915120	66048	1.00e−0	5.74e−3	9.99e−1	7.80e−3
			random	915120	66048	1.00e−0	1.11e−2	1.00e−0	2.12e−2
	3	*BNet*	prefix	66608	66048	9.99e−1	9.83e−3	1.00e−0	1.01e−2
			random	66608	66048	1.00e−0	1.66e−2	1.00e−0	2.75e−2
		IBNet	prefix	275504	66048	9.98e−1	6.84e−3	9.97e−1	9.00e−3
			random	275504	66048	1.00e−0	1.21e−2	1.00e−0	3.55e−2

in square-sum-layer. More accurate approximation is achieved using square-sum-layer comparing to dense-dense-layer, which is due to fact that the functionals can be exactly represented by the former task layer but not the latter. The post-training relative errors for *Butterfly-net* and *Inflated-Butterfly-net* given the same initialization, L_ξ, and task layer, remain similar in all cases. Hence the significant larger number of parameters in *Inflated-Butterfly-net* does not improve the post-training accuracy much. Most importantly, as we compare the post-training relative errors of different initializations under dense-dense-layer, *Butterfly* initialized networks achieves better accuracy comparing to its random initialized counterpart. The dense-dense-layers here are all randomly initialized and only the weights in *Butterfly-net* and *Inflated-Butterfly-net* are initialized differently. Hence we conclude that *Butterfly* initialization, even just on part of the whole neural network, helps finding better approximations in representing functionals (5.2) and (5.3).

6 Conclusion and discussion

A low-complexity convolutional neural network with structured *Butterfly* initialization and sparse cross-channel connections is proposed, motivated by the *Butterfly* scheme. The functional representation by *Butterfly-net* is optimal in the sense that the model complexity is $\mathcal{O}(K\log N)$ and the computational complexity is $\mathcal{O}(N\log N)$ for N and K being the input and output vector lengths. The approximation accuracy to the Fourier kernel is proved to exponentially decay as the depths of the *Butterfly-net* increases. We also conduct an approximation analysis of *Butterfly-net* in representing a large class of problems in scientific computing and image and signal processing. Comparing *Butterfly-net* to fully connected networks, the leading term in network complexity is reduced from $\varepsilon^{-N/s}$ down to $\varepsilon^{-K/s}$, where N is the input dimension, K is the effective dimension, and s is regularization level of the problem. Regular CNN can be viewed a special network under the analysis.

The trained *Butterfly-net*s from *Butterfly* initialization and random initialization are applied to represent discrete Fourier transforms and energy functionals. For these examples, *Butterfly-net* achieves better accuracy than its no-trained version. We also compared *Butterfly-net* against *Inflated-Butterfly-net*. From the numerical results, we find that *Butterfly-net* is able to achieve similar accuracy as *Inflated-Butterfly-net*, while the number of parameters is orders of magnitudes smaller. In the transfer learning settings, *Butterfly-net* generalizes better than *Inflated-Butterfly-net* when the distribution of the input data has domain shift.

The work can be extended in several directions. First, more applications of the *Butterfly-net* can be explored such as those in image analysis and signal processing. Likely, *Butterfly-net* is able to replace some CNN structures in practice such that similar accuracy can be achieved while the parameter number is much reduced. Second, our current theoretical analysis does not address the case when the input data contain noise. In particular, adding rectified layers in *Butterfly-net* can be interpreted as a thresholding denoising operation applied to the intermediate representations; a statistical analysis is desired.

Acknowledgments

The work of YL and JL is supported in part by National Science Foundation via grants DMS-1454939, DMS-2012286 and ACI-1450280. XC is partially supported by NSF (DMS-1818945, DMS-1820827), NIH (grant R01GM131642) and the Alfred P. Sloan Foundation. We thank the anonymous referees for constructive suggestions.

A Proof of Theorem 2.1

Here we first include a well-known lemma of Chebyshev interpolation for completeness and then prove Theorem 2.1.

Lemma A.1. *Let $f(y)\in C_{[a,b]}$ and $\mathcal{P}_r$ be the space spanned by the monomials y^r. The projection operator Π_r mapping f into its Lagrange interpolation on the r Chebyshev grid obeys*

$$\|f-\Pi_r f\|_\infty \leq \left(2+\frac{2}{\pi}\ln r\right)\inf_{g\in\mathcal{P}_r}\|f-g\|_\infty. \tag{A.1}$$

The proof of Lemma A.1 can be found in [45].

Proof of Theorem 2.1. The Fourier kernel $\mathcal{K}(\xi,t)=e^{-2\pi\imath\xi\cdot t}$ can be decomposed as

$$\begin{aligned}\mathcal{K}(\xi,t)&=e^{-2\pi\imath(\xi\cdot t-\xi_0\cdot t-\xi\cdot t_0+\xi_0\cdot t_0)}\cdot e^{-2\pi\imath\xi_0\cdot t}\cdot e^{-2\pi\imath\xi\cdot t_0}\cdot e^{2\pi\imath\xi_0\cdot t_0}\\&=e^{-2\pi\imath R(\xi,t)}\cdot e^{-2\pi\imath\xi_0\cdot t}\cdot e^{-2\pi\imath\xi\cdot t_0}\cdot e^{2\pi\imath\xi_0\cdot t_0},\end{aligned} \tag{A.2}$$

where $R(\xi,t)=(\xi-\xi_0)\cdot(t-t_0)$, t_0 and ξ_0 are centers of B and A respectively.

Next, we show the r-term truncation error for the first term in the second line of (A.2). Based on the power expansion of $e^{-2\pi\imath R(\xi,t)}$, i.e.,

$$e^{-2\pi\imath R(\xi,t)}=\sum_{k=0}^{\infty}\frac{(-2\pi\imath R(\xi,t))^k}{k!}, \tag{A.3}$$

the r-term truncation error can be bounded as

$$\begin{aligned}\delta&=\left|e^{-2\pi\imath R(\xi,t)}-\sum_{k=0}^{r}\frac{(-2\pi\imath R(\xi,t))^k)}{k!}\right|=\left|\sum_{k=r+1}^{\infty}\frac{(-2\pi\imath R(\xi,t))^k}{k!}\right|\\&\leq\sum_{k=r+1}^{\infty}\frac{(\pi w(A)w(B))^k}{2^k k!}\leq\sum_{k=r+1}^{\infty}\left(\frac{\pi e w(A)w(B)}{2k}\right)^k\leq\left(\frac{\pi e w(A)w(B)}{2r}\right)^r,\end{aligned} \tag{A.4}$$

where the last inequality uses $w(A)w(B)\leq\frac{r}{\pi e}$. We also notice that, for any fixed ξ, $\sum_{k=0}^{r}\frac{(2\pi\imath R(\xi,\cdot))^k}{k!}\in\mathcal{P}_r$. Applying Lemma A.1, we obtain

$$\left\|e^{-2\pi\imath R(\xi,t)}-\sum_{k=1}^{r}e^{-2\pi\imath R(\xi,t_k)}\mathcal{L}_k(t)\right\|_\infty\leq\left(2+\frac{2}{\pi}\ln r\right)\delta. \tag{A.5}$$

By substituting the explicit expression of $R(\xi,t)$, we obtain one of the conclusion,

$$\left\|e^{-2\pi\imath\xi\cdot t}-\sum_{k=1}^{r}e^{-2\pi\imath\xi\cdot t_k}e^{-2\pi\imath\xi_0\cdot(t-t_k)}\mathcal{L}_k(t)\right\|_\infty\leq\left(2+\frac{2}{\pi}\ln r\right)\left(\frac{\pi e w(A)w(B)}{2r}\right)^r, \tag{A.6}$$

for any $\xi\in A$ and $t\in B$.

Similarly, for any fixed t, we have $\sum_{k=0}^{r}\frac{(2\pi\imath R(\xi,t))^k}{k!}\in\mathcal{P}_r(\xi)$. Hence the second conclusion can be obtained through the same procedure. □

B Complex valued operation with ReLU activation function

A complex-valued linear operator can be represented through real-valued operation with ReLU activation function. Assume we want to represent the complex-valued linear operation

$$y = ax, \tag{B.1}$$

where $x, y \in \mathbb{C}$ can be generalized to vectors and $a \in \mathbb{C}$ can be generalized to a complex-valued matrix. We first introduce the real-valued representation for complex numbers/vectors. A complex number $x = \Re x + \imath \Im x \in \mathbb{C}$ is represented as

$$\begin{pmatrix} (\Re x)_+ & (\Im x)_+ & (\Re x)_- & (\Im x)_- \end{pmatrix}^\top, \tag{B.2}$$

where $(z)_+ = \max(z,0)$ and $(z)_- = -\min(z,0)$ for any $z \in \mathbb{R}$. If x is a complex vector of size n, then the real-valued representation concatenates (B.2) for each complex value and results a real vector of size $4n$.

The multiplication $y = ax$ is produced as the ReLU activation function acting on a matrix vector multiplication, i.e.,

$$\sigma\left(\begin{pmatrix} \Re a & -\Im a & -\Re a & \Im a \\ \Im a & \Re a & -\Im a & -\Re a \\ -\Re a & \Im a & \Re a & -\Im a \\ -\Im a & -\Re a & \Im a & \Re a \end{pmatrix}\begin{pmatrix} (\Re x)_+ \\ (\Im x)_+ \\ (\Re x)_- \\ (\Im x)_- \end{pmatrix}\right) = \begin{pmatrix} (\Re y)_+ \\ (\Im y)_+ \\ (\Re y)_- \\ (\Im y)_- \end{pmatrix}, \tag{B.3}$$

where the complex-valued linear operator a is extended to be a real-valued 4×4 matrix. Extension of a to a matrix can also be done through concatenating in both row and column directions.

In order to simplify the description in this paper, we define an extensive assign operator as $\overset{\diamond}{=}$ such that the 4 by 4 matrix A in (B.3) then obeys $A \overset{\diamond}{=} a$.

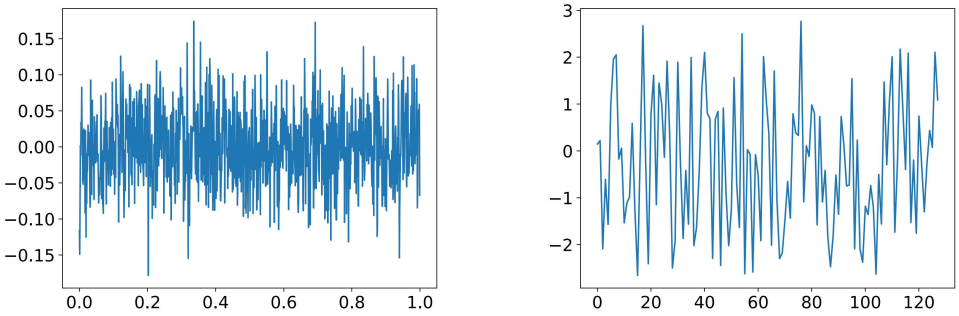
The above approach of using ReLU activation function is also adopted in [49]. We discuss the usage of other activation functions in the main text.

C Example signals of datasets

We provide instance input and output data for each dataset in Table 4, namely Fig. 6 for DFT-Lfreq, Fig. 7 for DFT-Hfreq, Fig. 8 for DFTSmooth-Lfreq, and Fig. 9 for DFTSmooth-Hfreq.

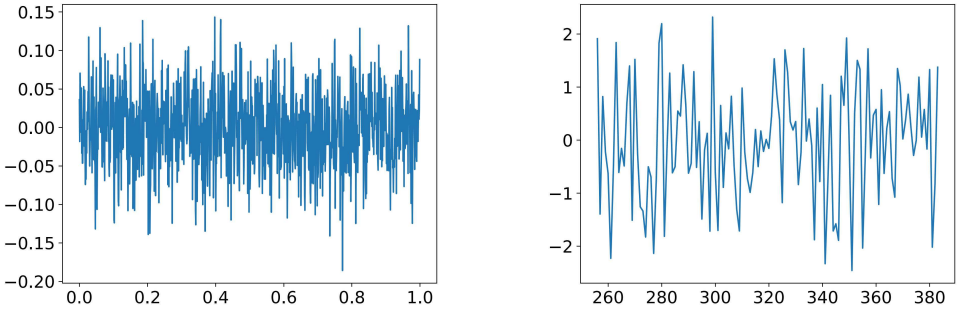
D Convergence behaviors

We provide four convergence behaviors for the training loss of *Butterfly-net* and *Inflated-Butterfly-net* with random and *Butterfly* initialization applied to DFTSmooth-Lfreq in Fig. 10. In all four networks, the number of layers after switch layer is $L_\xi = 1$. Other settings have the similar convergence behaviors.



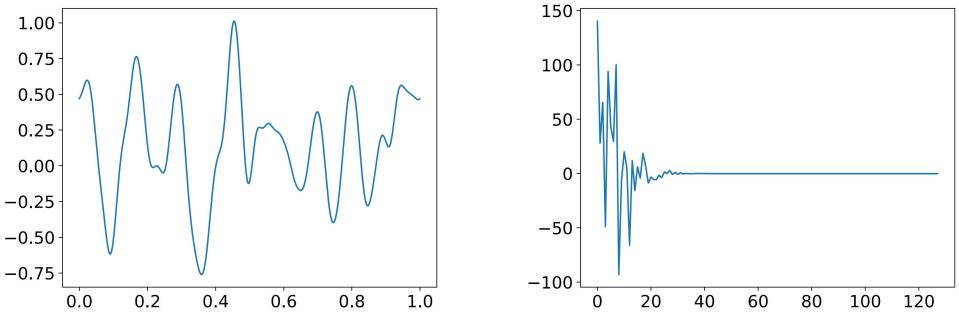
(a) Input data $\vec{x}$ (b) Real part of output data $\vec{y}$

Figure 6: One instance in dataset DFT-Lfreq.



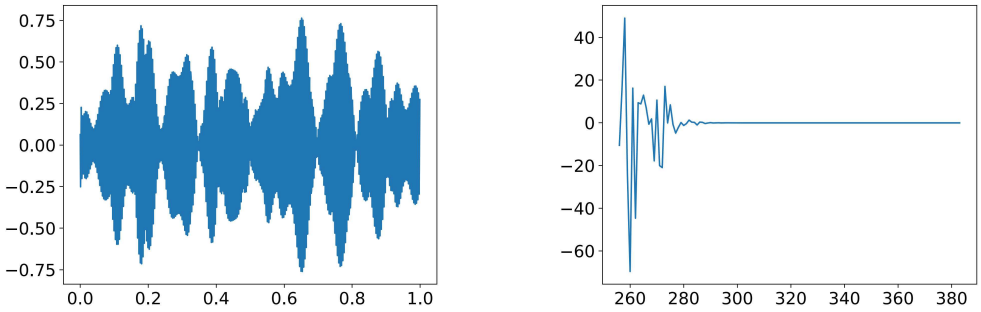
(a) Input data $\vec{x}$ (b) Real part of output data $\vec{y}$

Figure 7: One instance in dataset DFT-Hfreq.



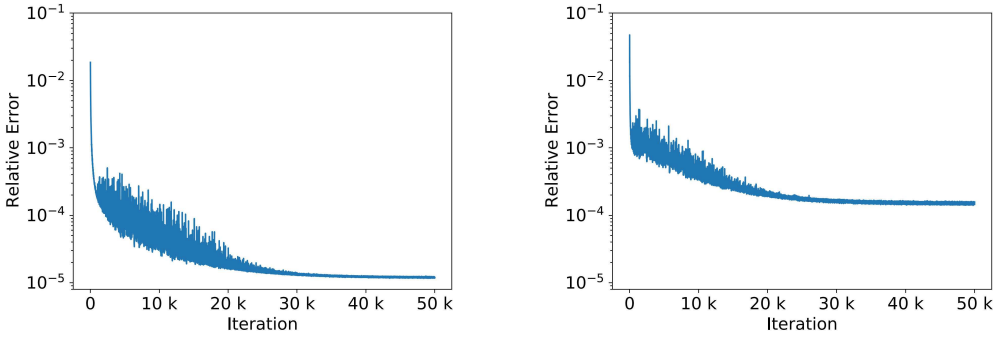
(a) Input data $\vec{x}$ (b) Real part of output data $\vec{y}$

Figure 8: One instance in dataset DFTSmooth-Lfreq.

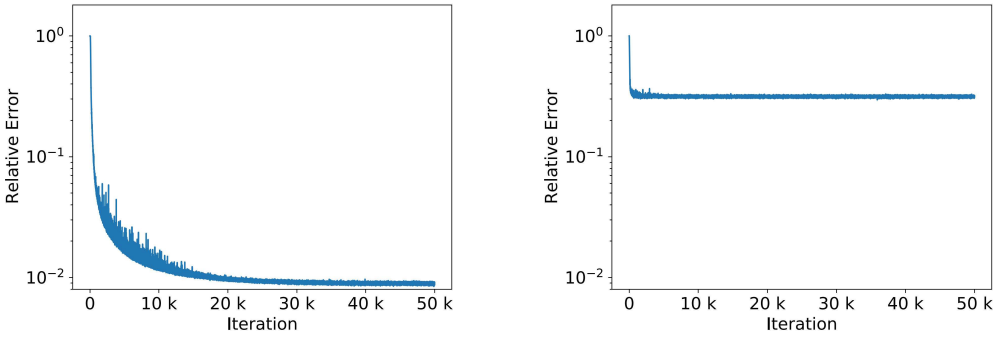


(a) Input data $\vec{x}$ (b) Real part of output data $\vec{y}$

Figure 9: One instance in dataset DFTSmooth-Hfreq.



(a) *Butterfly-net* with *Butterfly* initialization (b) *Inflated-Butterfly-net* with *Butterfly* initialization



(c) *Butterfly-net* with random initialization (d) *Inflated-Butterfly-net* with random initialization

Figure 10: Convergence behavior for various networks applied to DFTSmooth-Lfreq dataset. The number of layers after switch layer for all networks is $L_\xi = 1$.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] C. Bao, Q. Li, Z. Shen, C. Tai, L. Wu, and X. Xiang. Approximation analysis of convolutional neural networks, 2019.

[3] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[4] J. Behler and M. Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical review letters*, 98(14):146401, 2007.

[5] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. *Nature*, 521(7553):436–444, 2015.

[6] J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28 – 41, 2018. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.06.056. URL `http://www.sciencedirect.com/science/article/pii/S092523121830794X`.

[7] H. Bölcskei, P. Grohs, G. Kutyniok, and P. Petersen. Optimal approximation with sparsely connected deep neural networks. *SIAM Journal on Mathematics of Data Science*, 1(1):8–45, 2019. doi: 10.1137/18M118709X. URL `https://doi.org/10.1137/18M118709X`.

[8] J.-F. Cai, B. Dong, S. Osher, and Z. Shen. Image restoration: total variation, wavelet frames, and beyond. *Journal of the American Mathematical Society*, 25(4):1033–1089, 2012.

[9] E. J. Candès, L. Demanet, and L. Ying. Fast computation of Fourier integral operators. *SIAM J. Sci. Comput.*, 29(6):2464–2493, jan 2007. URL `http://epubs.siam.org/doi/abs/10.1137/060671139`.

[10] E. J. Candès, L. Demanet, and L. Ying. A fast butterfly algorithm for the computation of Fourier integral operators. *Multiscale Model. Simul.*, 7(4):1727–1750, jan 2009. URL `http://epubs.siam.org/doi/abs/10.1137/080734339`.

[11] T. F. Chan and J. J. Shen. *Image processing and analysis: variational, PDE, wavelet, and stochastic methods*, volume 94. SIAM, 2005.

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

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

[14] O. Delalleau and Y. Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

[15] L. Demanet and L. Ying. Discrete symbol calculus. *SIAM Rev.*, 53(1):71–104, jan 2011. URL `http://epubs.siam.org/doi/10.1137/080731311`.

[16] L. Demanet, M. Ferrara, N. Maxwell, J. Poulson, and L. Ying. A butterfly algorithm for synthetic aperture radar imaging. *SIAM Journal on Imaging Sciences*, 5(1):203–243, jan 2012. URL `http://epubs.siam.org/doi/10.1137/100811593`.

[17] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[18] R. Eldan and O. Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.
[19] Y. Fan and L. Ying. Solving electrical impedance tomography with deep learning, jun 2019. http://arxiv.org/abs/1906.03944.
[20] Y. Fan, L. Lin, L. Ying, and L. Zepeda-Núñez. A multiscale neural network based on hierarchical matrices, 2018. https://arxiv.org/abs/1807.01883.
[21] Y. Fan, J. Feliu-Fabà, L. Lin, L. Ying, and L. Zepeda-Núñez. A multiscale neural network based on hierarchical nested bases. *Res. Math. Sci.*, 6(2):21, mar 2019a. doi: 10.1007/s40687-019-0183-3.
[22] Y. Fan, C. Orozco Bohorquez, and L. Ying. BCR-Net: A neural network based on the nonstandard wavelet form. *J. Comput. Phys.*, 384:1–15, may 2019b. doi: 10.1016/J.JCP.2019.02.002.
[23] J. Feliu-Faba, Y. Fan, and L. Ying. Meta-learning pseudo-differential operators with deep neural networks, jun 2019. http://arxiv.org/abs/1906.06782.
[24] A. R. Gallant and H. White. There exists a neural network that does not make avoidable mistakes. In *Proceedings of the Second Annual IEEE Conference on Neural Networks, San Diego, CA, I*, 1988.
[25] J. He and J. Xu. MgNet: A unified framework of multigrid and convolutional neural network. *Science China Mathematics*, 62(7):1331–1354, jul 2019. doi: 10.1007/s11425-019-9547-2. URL `http://link.springer.com/10.1007/s11425-019-9547-2`.
[26] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
[27] Y. Khoo and L. Ying. SwitchNet: a neural network model for forward and inverse scattering problems, oct 2018. http://arxiv.org/abs/1810.09675.
[28] Y. Khoo, J. Lu, and L. Ying. Solving for high-dimensional committor functions using artificial neural networks. *Research in the Mathematical Sciences*, 6(1):1, 2018. ISSN 2197-9847. doi: 10.1007/s40687-018-0160-2. URL `https://doi.org/10.1007/s40687-018-0160-2`.
[29] S. Kunis and I. Melzer. A stable and accurate butterfly sparse Fourier transform. *SIAM J. Numer. Anal.*, 50(3):1777–1800, jan 2012. URL `http://epubs.siam.org/doi/10.1137/110839825`.
[30] N. Le Roux and Y. Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.
[31] Y. Li and H. Yang. Interpolative butterfly factorization. *SIAM J. Sci. Comput.*, 39(2):A503–A531, 2017. URL `http://dx.doi.org/10.1137/16M1074941`.
[32] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying. Butterfly factorization. *Multiscale Model. Simul.*, 13(2):714–732, jan 2015a. URL `http://epubs.siam.org/doi/10.1137/15M1007173`.
[33] Y. Li, H. Yang, and L. Ying. A multiscale butterfly algorithm for multidimensional Fourier integral operators. *Multiscale Model. Simul.*, 13(2):1–18, jan 2015b. URL `http://epubs.siam.org/doi/10.1137/140997658http://arxiv.org/abs/1411.7418`.
[34] Y. Li, H. Yang, and L. Ying. Multidimensional butterfly factorization. *Applied and Computational Harmonic Analysis*, 44(3):737–758, may 2018. doi: 10.1016/J.ACHA.2017.04.002. URL `https://www.sciencedirect.com/science/article/pii/S1063520317300271`.
[35] Y. Li, J. Lu, and A. Mao. Variational training of neural network approximations of solution maps for physical models, may 2019. http://arxiv.org/abs/1905.02789.
[36] S. Liang and R. Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.
[37] Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-net: Learning PDEs from data. 80:3208–3216, 2018. URL `http://proceedings.mlr.press/v80/long18a.html`.

[38] J. Lu, Z. Shen, H. Yang, and S. Zhang. Deep network approximation for smooth functions, jan 2020. http://arxiv.org/abs/2001.03040.
[39] S. Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.
[40] H. Mhaskar, Q. Liao, and T. Poggio. Learning functions: when is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.
[41] H. N. Mhaskar and T. Poggio. Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, 14(06):829–848, 2016.
[42] E. Michielssen and A. Boag. A multilevel matrix decomposition algorithm for analyzing scattering from large structures. *IEEE Trans. Antennas Propag.*, 44(8):1086–1093, 1996. URL `http://ieeexplore.ieee.org/document/511816/`.
[43] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.
[44] M. O'Neil, F. Woolfe, and V. Rokhlin. An algorithm for the rapid evaluation of special function transforms. *Appl. Comput. Harmon. Anal.*, 28(2):203–226, 2010.
[45] T. J. Rivlin. *Chebyshev polynomials: from approximation theory to algebra and number theory*. Wiley-Interscience, 2nd edition, 1990.
[46] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Med. Image Comput. Comput. Interv.*, volume 9351, pages 234–241. Springer Verlag, 2015. ISBN 9783319245737. doi: 10.1007/978-3-319-24574-4_28.
[47] E. Schneider, L. Dai, R. Q. Topper, C. Drechsel-Grau, and M. E. Tuckerman. Stochastic neural network approach for learning high-dimensional free energy surfaces. *Physical review letters*, 119(15):150601, 2017.
[48] Matus Telgarsky. Benefits of depth in neural networks. 49:1517–1539, 2016. URL `http://proceedings.mlr.press/v49/telgarsky16.html`.
[49] Z. Xu, Y. Li, and X. Cheng. Butterfly-Net2: Simplified butterfly-net and Fourier transform initialization, dec 2019. http://arxiv.org/abs/1912.04154.
[50] D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, oct 2017a. ISSN 18792782. doi: 10.1016/j.neunet.2017.07.002.
[51] D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017b.
[52] L. Ying. Sparse Fourier transform via butterfly algorithm. *SIAM J. Sci. Comput.*, 31(3):1678–1694, jan 2009. URL `http://epubs.siam.org/doi/10.1137/08071291X`.
[53] L. Zhang, J. Han, H. Wang, R. Car, and W. E. Deepcg: Constructing coarse-grained models via deep neural networks. *The Journal of Chemical Physics*, 149(3):034101, 2018. doi: 10.1063/1.5027645. URL `https://doi.org/10.1063/1.5027645`.
[54] D.-X. Zhou. Universality of deep convolutional neural networks. *Applied and Computational Harmonic Analysis*, 2019. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2019.06.004. URL `http://www.sciencedirect.com/science/article/pii/S1063520318302045`.