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

Xiuyuan Cheng[♯], Yingzhou Li[♯⋆], Jianfeng Lu[♯†]

♯ Department of Mathematics, Duke University
† Department of Chemistry and Department of Physics, Duke University
⋆ yingzhou.li@duke.edu

August 22, 2019

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 across-channel connections, which aims at an optimal hierarchical function representation of the input signal. 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. Due to the ability of *Butterfly-net* to approximate Fourier and local Fourier transforms, the result can be used for approximation upper bound for CNNs in a large class of problems. The analytical results are validated by numerical experiments on the approximation of a 1D Fourier kernel and of the energy of 1D and 2D Poisson's equations. *Butterfly-net* with trained parameters outperforms the hard-coded *Butterfly-net* and achieves similar accuracy as the trained CNN but with much less parameters. In addition, better robustness of *Butterfly-net* against CNN is demonstrated when the distribution of the input data has domain shift.

1 Introduction

Deep network is a central tool in machine learning and data analysis nowadays [3]. In particular, Convolutional Neural Network (CNN) has been proved to be a powerful tool in image recognition and representation. Deep learning has also emerged to be successfully applied in solving PDEs [4, 26, 35] and physics problems [2, 15, 33, 43, 47], showing the potential of becoming a tool of great use for computational mathematics and physics as well. Given the wide application of PDE and wavelet based methods in image and signal processing [6, 9, 36], 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 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 [1, 11, 24] 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 networks, particular the superior performance of CNNs in image and audio signal processing.

Motivated by the empirical success, the approximation advantage of deep networks over shallow ones has been theoretically analyzed in several places. However, most results assume stacked fully connected layers and do not apply to CNN which has 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 [10, 37, 38], yet the network architecture differ from the standard CNN. The universal approximation theory of CNN for general functions has been studied in [48]. We review the related literature more below.

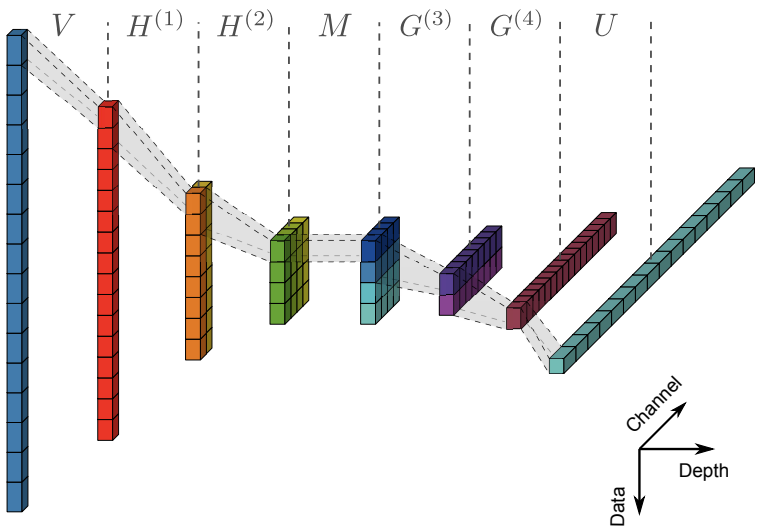


Figure 1: *Butterfly-net* in 1D. Layer 1 and 7 are 1D convolutional layers. Layer 2, 3, 5 and 6 are 1D convolutional layers with 2 non-mixing channels for each input channel. Layer 4 is a local mixing layer, which only involves local operations. $V, H^{(\ell)}, M, G^{(\ell)}, U$ indicate the matrix representation of the convolutional layers, c.f. Section 3.3. The name "butterfly" can be understood from the shape of the network.

This paper proposes a specific architecture of convolutional neural networks based on the *Butterfly* scheme originally developed for the fast computation of special function transforms [39, 41, 46] and Fourier integral operators [7, 8, 29, 30, 31, 32]. *Butterfly* algorithm provides a hierarchical structure with locally low-rank interpolation of kernel functions, and can be applied to solve many PDE problems. In terms of computational complexity, the scheme is near optimal for Fourier kernels and a large class of Fourier integral operators. The proposed *Butterfly-net* explicitly adopts the hierarchical structure in *Butterfly* algorithms as the stacked convolutional layers. If the parameters are hard-coded as that in the *Butterfly* algorithm (*Butterfly* coefficients), then *Butterfly-net* collectively computes the Fourier coefficients of the input signal with guaranteed numerical accuracy. Unlike regular CNN which is fully connected across the unordered channels, the channels in the *Butterfly-net* has a clear correspondence with the frequency band, namely the position in the spectral representation of the signal, and meanwhile, the inter-channel weights are sparsely connected. *Butterfly-net* is much lighter than the usual CNN: the model complexity (in terms of number of parameters in the parametrization) is $O(N)$ and computational complexity is $O(N \log N)$, where N is the length of the discrete input signal.

The approximation error of *Butterfly-net* 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. Our theoretical result provides an approximation upper bound of the CNNs which are trained in practice, an exemplar case of which is shown in Section 5.

Butterfly-net can also be trained like other network structures. Numerically, we show that *Butterfly-net* trained with *Butterfly* coefficient initialization achieves better approximation error comparing to the no-trained *Butterfly-net* in the given data distribution. The comparison to CNN has also been studied. When both networks trained with the *Butterfly* coefficient initialization, *Butterfly-net* is able to achieve comparable accuracy while the number of parameters is much smaller. Also *Butterfly-net* has better robustness when the distribution of the input data has domain shift.

1.1 Contributions

The *Butterfly-net*, a low-complexity CNN, motivated by the *Butterfly* algorithm, is proposed with the following properties.

1) The network has the structure of stacked convolutional layers and sparse across-channel connections.

The channel indexing is in the order of the frequency band, and the intermediate representations in the network have the interpretation of local Fourier transforms.

2) The networks gives a near-optimal hierarchical representation of the Fourier kernel, with model complexity $O(N)$ and computational complexity $O(N \log N)$.

3) The approximation error of the Fourier Kernel is theoretically proved to be exponentially decay as the depth increases. Combined with fully-connected layers, the approximation analysis of *Butterfly-net* provides an approximation upper bound of CNNs in a large class of problems in PDEs and image and signal processing.

4) Numerically, we observe that the trained *Butterfly-net* outperforms the traditional *Butterfly* algorithm in representing both Fourier transforms and energy functionals of elliptic PDEs.

5) In the comparison of trained *Butterfly-net* with its CNN counterpart (i.e., *Butterfly-net* structure with fully connected channels), we observe that they achieve similar accuracy while *Butterfly-net* contains much less number of parameters. *Butterfly-net* also admits better transfer learning capability when the distribution of the input data has domain shift.

1.2 Related Works

Before we explain all these in more detail 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 [18, 19], which are more suitable in training smooth linear or nonlinear operator due to the nature of $\mathcal{H}$-matrices. In addition to that, nonstandard wavelet form inspired the design of BCR-Net [20], 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 [17] and pseudo-differential operator [21]. Multigrid method also inspired MgNet [23]. In addition to the above approximation of relatively smooth operators, *Butterfly* algorithm [8, 31, 30, 29] inspired the design of Switch-Net [25], which is a non-convolutional three layer *Butterfly* structure 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 [11, 24] showing that such network 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 [24], when exponentially many hidden neurons are used. Gallant and White [22] 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 [1] showed that the mean squared error of the approximation, integrated with arbitrary data distribution on the sphere, decays as $O(n^{-1})$ 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 [22] and [1], it is different from the hierarchical function representation scheme as what we consider here.

Approximation power of deep networks. The expressive power of deep networks has drawn many research interests in recent years. The approximation power of multi-layer Restricted Boltzmann Machines (RBM) was studied in [28], 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. [5] derived lower bounds for the uniform approximation of square-integrable functions, and proved the asymptotic optimality of the sparsely connected deep 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 [12] 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 [40, 44]. Eldan and Shamir [16] constructed a concrete target function which distinguishes three and two-layer networks. Liang and Srikant [34] 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 [45] in the context of approximation in Sobolev spaces. The above works address deep networks with fully-connected layers, instead of having geometrically-constrained constructions like CNNs.

Deep networks with these constraints are relatively less analyzed. The approximation power of a hierarchical binary tree network was studied in [37, 38] which supports the potential advantage of deep CNNs. Cohen et al. [10] 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 [48] proposed the universal approximation theory of deep CNN with a tight estimation on the number of free parameters. Comparing to [48], our network has sparse connection in channel direction and the function class in this work is more restricted, hence the approximation bound admits exponential decay in depth.

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 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* with *Butterfly* coefficient initialization together with trained *Butterfly-net* are presented in Section 5 including the comparison to CNNs. 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 discrete Fourier kernel, which is crucial to the efficiency of *Butterfly* algorithm and *Butterfly-net*. Then in Section 2.2, we reviews the *Butterfly* algorithm for Fourier kernel. Readers, who are familiar with butterfly algorithm, should be save to skip it.

2.1 Low-rank Approximation of Fourier Kernel

Fourier kernel throughout this paper is defined as,

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

where K denotes the range of frequency of interests. It is well-known that the discrete Fourier transform (DFT) matrix, i.e., uniform discretization of (1) with proper scaling, has orthonormal rows and columns. Hence, the DFT matrix is a full rank matrix, so is the Fourier kernel. In order to obtain the low-rank property, we first define the L level hierarchical partition of $[0, 1)$ and $[-K/2, K/2)$, respectively. Let $B_0^0 = [0, 1)$ be the domain on level 0. On level 1, the domain B_0^0 is evenly partitioned into $B_0^1 = [0, 1/2)$ and $B_1^1 = [1/2, 1)$. 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 level ℓ is $\{B_j^\ell, j = 0, \ldots, 2^\ell - 1\}$ and each $B_j^\ell = [j/2^\ell, j+1/2^\ell)$ is of length $2^{-\ell}$. Similarly, we conduct the hierarchical bipartition on $[-K/2, K/2)$, denoted them as A_i^ℓ, and A_i^ℓ is of length $K \cdot 2^{-\ell}$. As we shall see, the partition of domain will be used in a complementary way, A_i^ℓ will be paired with $B_j^{L-\ell}$ for all choices of i and j, such that the Fourier kernel restricted to A_i^ℓ and $B_j^{L-\ell}$ permits a low-rank approximation, see Theorem 2.1. Theorem 2.1 actually depends only on the length of the domains of t and ξ, but does not depend on the location.

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

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

The r points Chebyshev interpolation of any function $f(x)$ on $[-1/2, 1/2]$ is defined as,

$$\Pi_r f(x) = \sum_{k=1}^{r} f(z_k) \mathcal{L}_k(x), \tag{3}$$

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

$$\mathcal{L}_k(x) = \prod_{p \neq k} \frac{x - z_p}{z_k - z_p}. \tag{4}$$

Several earlier works [7, 8, 31] 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 L and r be two parameters such that $\pi e K \leqslant r 2^L$. For any level $\ell = 0, 1, \ldots, L$, let A^ℓ and $B^{L-\ell}$ denote two connected subdomains of $[-K/2, K/2)$ and $[0, 1)$ with length $K \cdot 2^{-\ell}$ and $2^{\ell - L}$ respectively. Then for any $\xi \in A^\ell$ and $t \in B^{L-\ell}$, there exists a Chebyshev interpolation representation of the Fourier kernel,*

$$\left| e^{-2\pi \imath \xi \cdot t} - \sum_{k=1}^{r} e^{-2\pi \imath \xi \cdot t_k} e^{-2\pi \imath \xi_0 \cdot (t - t_k)} \mathcal{L}_k(t) \right| \leqslant \left(2 + \frac{2}{\pi} \ln r \right) \left(\frac{2\pi e K}{4 r 2^L} \right)^r, \tag{5}$$

and

$$\left| e^{-2\pi \imath \xi \cdot t} - \sum_{k=1}^{r} e^{-2\pi \imath (\xi - \xi_k) \cdot t_0} \mathcal{L}_k(\xi) e^{-2\pi \imath \xi_k \cdot t} \right| \leqslant \left(2 + \frac{2}{\pi} \ln r \right) \left(\frac{2\pi e K}{4 r 2^L} \right)^r, \tag{6}$$

where ξ_0 and t_0 are the centers of A^ℓ and $B^{L-\ell}$ respectively, ξ_k and t_k are Chebyshev points on A^ℓ and $B^{L-\ell}$ respectively.

The proof of Theorem 2.1 is available in Appendix A.

At ℓ-th level, we know that A_i^ℓ and $B_j^{L-\ell}$ satisfy the assumption in Theorem 2.1 for any $i = 0, \ldots, 2^\ell - 1$ and $j = 0, \ldots, 2^{L-\ell} - 1$. Hence the Fourier kernel restricted to any complementary pair of domains A_i^ℓ and $B_j^{L-\ell}$ is low-rank. This property is also known as complementary low-rank property [30, 31].

2.2 Butterfly Algorithm for Fourier Kernel

This section briefly describes the *Butterfly* algorithm for Fourier kernel based on Theorem 2.1. Given a function discretized on a uniform grid, $\{f(t_q), t_q \in [0, 1)\}$, the goal is to compute the discrete Fourier transform $\{\hat{f}(\xi_p), \xi_p \in [-K/2, K/2)\}$ defined by

$$\hat{f}(\xi_p) = \sum_{t_q \in [0,1)} \mathcal{K}(\xi_p, t_q) f(t_q), \quad \xi_p \in [-\frac{K}{2}, \frac{K}{2}). \tag{7}$$

Recall that at level ℓ for any pair of intervals A_i^ℓ and $B_j^{L-\ell}$, the Fourier kernel obeying the complementary low-rank property and hence the submatrix $\{\mathcal{K}(\xi_p, t_q)\}_{\xi_p \in A_i^\ell, t_q \in B_j^{L-\ell}}$ is approximately of a constant rank. An explicit formula for the low-rank approximation is given by discrete version of Theorem 2.1. We summarize the *Butterfly* algorithm for Fourier kernel as follows. Without loss of generality, we assume L is even.

1. *Preliminaries.* Construct an L level hierarchical bipartition of both domains.

2. *Initialization (level $\ell = 0$).* For $A = A_0^0$ and each interval $B = B_j^L$, conduct a coefficient transference from uniform grid in B to Chebyshev points in B and denote the transferred coefficients as $\{\lambda_k^{AB}\}_{1\leqslant k\leqslant r}$, i.e.,

$$\lambda_k^{AB} = \sum_{t\in B} e^{-2\pi\imath\xi_0\cdot(t-t_k)}\mathcal{L}_k(t)f(t), \quad 1\leqslant k\leqslant r, \tag{8}$$

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

$$\hat{f}(\xi_p) = \sum_{B=\{B_j^L\}}\sum_{t_q\in B} e^{-2\pi\imath\xi_p\cdot t_q}f(t_q) \approx \sum_{B=\{B_j^L\}}\sum_{t_k\in B} e^{-2\pi\imath\xi_p\cdot t_k}\lambda_k^{AB} \quad \xi_p\in A, \tag{9}$$

where t_q and ξ_p denote uniform grid points and the approximation accuracy is controlled by r and L based on (5).

3. *Recursion (level $\ell = 1,\ldots,L/2$).* For each pair $(A,B) = (A_i^\ell, B_j^{L-\ell})$, construct the transferred coefficients $\{\lambda_k^{AB}\}_{1\leqslant k\leqslant r}$. Let $P = A_{\lfloor i/2\rfloor}^{\ell-1}$ denote the parent of A and $C = B_{2j}^{L-\ell+1}$ or $B_{2j+1}^{L-\ell+1}$ denote a child of B. Throughout, we shall use the notation $C\succ B$ when C is a child of B. At level $\ell-1$, the coefficients $\{\lambda_s^{PC}\}_{1\leqslant s\leqslant r}$ satisfy,

$$\hat{f}(\xi_p) \approx \sum_{C=\{B_j^{L-\ell+1}\}}\sum_{t_s\in C} e^{-2\pi\imath\xi_p\cdot t_s}\lambda_s^{PC} = \sum_{B=\{B_j^{L-\ell}\}}\sum_{\substack{C\succ B\\ t_s\in C}} e^{-2\pi\imath\xi_p\cdot t_s}\lambda_s^{PC} \quad \xi_p\in P. \tag{10}$$

Since $A\subset P$, the above approximation holds for $\xi_p\in A$ 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 $\{\lambda_k^{AB}\}_{1\leqslant k\leqslant r}$, i.e.,

$$\lambda_k^{AB} = \sum_{\substack{C\succ B\\ t_s\in C}} e^{-2\pi\imath\xi_0\cdot(t_s-t_k)}\mathcal{L}_k(t_s)\lambda_s^{PC}, \quad 1\leqslant k\leqslant r, \tag{11}$$

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

$$\hat{f}(\xi_p) \approx \sum_{B=\{B_j^{L-\ell}\}}\sum_{t_k\in B} e^{-2\pi\imath\xi_p\cdot t_k}\lambda_k^{AB} \quad \xi_p\in A. \tag{12}$$

4. *Switch (level $\ell = L/2$).* For the levels visited, the Chebyshev interpolation is applied in variable t, while the interpolation is applied in variable ξ for levels $\ell > L/2$. Hence, we switch the interpolation method at this step. For all pairs $(A,B) = (A_i^{L/2}, B_j^{L/2})$ 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,

$$\lambda_k^{AB} := \sum_{s=1}^{r} e^{-2\pi\imath\xi_k^A\cdot t_s^B}\lambda_s^{AB} \approx \sum_{t_q\in B} e^{-2\pi\imath\xi_k^A\cdot t_q}f(t_q), \tag{13}$$

where t_q denotes the original uniform distributed points in B and the approximation is due to the definition of λ_s^{AB} and (6).

5. *Recursion (level $\ell = L/2+1,\ldots,L$).* Similar to the discussion in Step 3, for each pair $(A,B) = (A_i^\ell, B_j^{L-\ell})$, and the corresponding P and C, at level $\ell-1$, the coefficients $\{\lambda_s^{PC}\}_{1\leqslant s\leqslant r}$ satisfy,

$$\lambda_s^{PC} \approx \sum_{t_q\in C} e^{-2\pi\imath\xi_s\cdot t_q}f(t_q), \tag{14}$$

where ξ_s denotes the Chebyshev points in P. Given the second approximation in Theorem 2.1, we have, with notation ξ_p and ξ_k being uniform points and Chebyshev points in A respectively,

$$\begin{aligned}
\hat{f}(\xi_p) &= \sum_{B=\{B_j^{L-\ell}\}} \sum_{\substack{C>B \\ t_q\in C}} e^{-2\pi\imath\xi_p\cdot t_q} f(t_q) \\
&\approx \sum_{B=\{B_j^{L-\ell}\}} \sum_{\substack{C>B \\ t_q\in C}} \sum_{\xi_k\in A} e^{-2\pi\imath(\xi_p-\xi_k)\cdot t_0^C} \mathcal{L}_k(\xi_p) e^{-2\pi\imath\xi_k\cdot t_q} f(t_q) \\
&\approx \sum_{B=\{B_j^{L-\ell}\}} \sum_{\substack{C>B \\ t_q\in C}} \sum_{\xi_k\in A} e^{-2\pi\imath(\xi_p-\xi_k)\cdot t_0^C} \mathcal{L}_k(\xi_p) \left(\sum_{\xi_s\in P} e^{-2\pi\imath(\xi_k-\xi_s)\cdot t_0^C} \mathcal{L}_s(\xi_k) e^{-2\pi\imath\xi_s\cdot t_q} \right) f(t_q) \\
&\approx \sum_{B=\{B_j^{L-\ell}\}} \sum_{\xi_k\in A} e^{-2\pi\imath(\xi_p-\xi_k)\cdot t_0^C} \mathcal{L}_k(\xi_p) \left(\sum_{\xi_s\in P} \sum_{C>B} e^{-2\pi\imath(\xi_k-\xi_s)\cdot t_0^C} \mathcal{L}_s(\xi_k) \lambda_s^{PC} \right),
\end{aligned} \tag{15}$$

where t_0^C denotes the center of C, the first and second approximation are due to (6) and the last approximation comes from the reorganization of summations and the definition of λ_s^{PC}. The summations in the bracket in the last line of (15) defines a transference between coefficients. Hence, λ_k^{AB} is defined as

$$\lambda_k^{AB} = \sum_{\xi_s\in P} \sum_{C>B} e^{-2\pi\imath(\xi_k-\xi_s)\cdot t_0^C} \mathcal{L}_s(\xi_k) \lambda_s^{PC}, \tag{16}$$

which naturally satisfies (14).

6. *Termination (level $\ell = L$).* Finally, $\ell = L$, for $B = B_0^0$ and each $A = A_i^L$, we approximate the $\hat{f}(\xi_p)$ for $\xi_p \in A$ as

$$\begin{aligned}
\hat{f}(\xi_p) &= \sum_{t_q\in B} e^{-2\pi\imath\xi_p\cdot t_q} f(t_q) \\
&\approx \sum_{t_q\in B} \sum_{\xi_k\in A} e^{-2\pi\imath(\xi_p-\xi_k)\cdot t_0} \mathcal{L}_k(\xi_p) e^{-2\pi\imath\xi_k\cdot t_q} f(t_q) \approx \sum_{\xi_k\in A} e^{-2\pi\imath(\xi_p-\xi_k)\cdot t_0} \mathcal{L}_k(\xi_p) \lambda_k^{AB}.
\end{aligned} \tag{17}$$

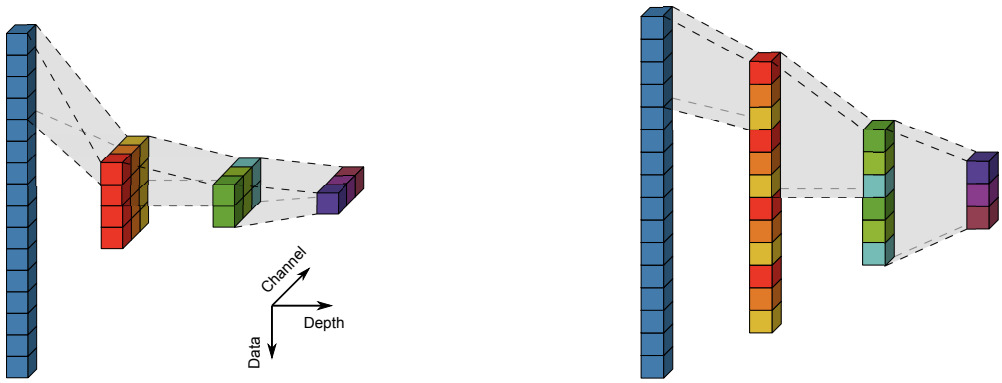
We notice that summation kernels in (8) and (11) relies only on the relative distance of ts and, hence, it can be viewed as a convolution kernel. Similarly summation kernels in (16) and (17) relies only on the relative distance of ξs, which can be viewed as convolution kernels as well. Such an observation 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 Figure 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 form the *Butterfly-net*. Finally, in Section 3.3, we provide the matrix representation of the *Butterfly-net*, which significantly simplifies the analysis in the next section. For simplicity of the notation and indexing, we will limit the discussion to 1D signals, while the extension to 2D and higher dimensional data is straightforward via tensor product structures and operations.

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.



(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.

Let $\{f(i,k_1) \mid i = 0,\dots,n-1; k_1 = 1,\dots,c_1\}$ 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 W_{k_2,i,k_1} for $i = 0,\dots,w-1$, $k_1 = 1,\dots,c_1$, and $k_2 = 1,\dots,c_2$. The output of the convolutional layer, under these notations, is written as

$$g(j,k_2) = \sum_{\substack{0\leqslant i\leqslant w-1\\ 1\leqslant k_1\leqslant c_1}} W_{k_2,i,k_1} f(i+s(j-1),k_1), \tag{18}$$

where $s \geqslant 1$ is the stride size and $j = 0,\dots,(n-w)/s$ 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., $f[ic_1+k_1] = f(i,k_1)$, $g[jc_2+k_2] = g(j,k_2)$, and $W[k_2, iw+k_1] = W_{k_2,i,k_1}$. Hence (18) can be represented as the matrix vector product,

$$g(j,:) = g[jc_2 + (1:c_2)] = Wf[sc_1 j + (1:wc_1)], \tag{19}$$

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 (coefficient transference in *Butterfly* algorithm, e.g., (8), (11), (16) and (17)) can be naturally represented as a multi-channel convolution. In this setting, unfolding channels is more natural. Let $B_0, B_1, \dots, B_{J-1}$ be a equal spaced partition of $[0,1)$, i.e., $B_j = [j/J, j+1/J)$, and $x_{k_1}^{B_j}$ denote the k_1-th discretization point in B_j for $k_1 = 1,\dots,c_1$. We further assume that the locations of $x_{k_1}^{B_j}$ relative to B_j are the same for all j. The input data is viewed as the function $f(x)$ evaluated at the points $x_{k_1}^{B_j}$, i.e., $f(j,k_1) = f(x_{k_1}^{B_j})$. Let $z_{k_2}^{B_j}$ be the interpolation points on B_j for $k_2 = 1,\dots,c_2$, with the Lagrange basis polynomial given by

$$\mathcal{L}_{k_2}(x) = \prod_{\substack{p=1\\ p\neq k_2}}^{c_2} \frac{x - z_p^{B_j}}{z_{k_2}^{B_j} - z_p^{B_j}}. \tag{20}$$

The interpolated function of $f(x)$ at $z_{k_2}^{B_j}$ is then defined as,

$$\begin{aligned} f(z_{k_2}^{B_j}) \approx g(j,k_2) = \sum_{k_1=1}^{c_1} \mathcal{L}_{k_2}(x_{k_1}^{B_j}) f(x_{k_1}^{B_j}) &= \sum_{k_1=1}^{c_1} \prod_{\substack{p=1 \\ p\neq k_2}}^{c_2} \frac{x_{k_1}^{B_j} - z_p^{B_j}}{z_{k_2}^{B_j} - z_p^{B_j}} f(j,k_1) \\ &= \sum_{k_1=1}^{c_1} \prod_{\substack{p=1 \\ p\neq k_2}}^{c_2} \frac{x_{k_1}^{B_0} - z_p^{B_0}}{z_{k_2}^{B_0} - z_p^{B_0}} f(j,k_1), \end{aligned} \tag{21}$$

where $j = 0, \ldots, J-1$. The last equality in (21) is due to the fact that each fraction in (21) depends only on the relative distance and is thus independent of B_j thanks to our assumption on the interpolation points. Therefore, we could denote $W_{k_2,i,k_1} = \prod_{p\neq k_2} {}^{(x_{k_1}^{B_0} - z_p^{B_0})}/_{(z_{k_2}^{B_0} - z_p^{B_0})}$, and thus the source transfer formula (21) can be interpreted as convolution (18) with the stride size being the same as the filter size, 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 interval B_j. Therefore, unfolding the channel index of both f and g leads to the natural ordering of the index of points on $[0, 1)$.

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

Without the assumption on $w = s$, the convolutional layer can also be understood as an interpolation with overlapping interval. Similar idea is used in Simpson's rule and multi-step methods.

3.2 *Butterfly-net* Architecture

This section formally introduce *Butterfly* algorithm inspired *Butterfly-net* architecture. We follow the exact structure of *Butterfly* algorithm in the introduction of *Butterfly-net*. 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 coefficients from *Butterfly* algorithm and an explanation related to the Fourier transform.

Let $f(t)$ be the input data viewed as a signal in time. Time-frequency analysis usually splits 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 will be analyzed separately and will not be mixed again. This motivates us to propose non-mixing channels in our *Butterfly-net*, since the channel has a correspondence with frequency modes in our setting. Assume that 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*, r denote the size mixing channels. Without loss of generality, we assume that L is an even integer such that $L \leqslant \log N$. All these settings are identical to that in *Butterfly* algorithm. Finally we propose the *Butterfly-net* architecture as follows (see Figure 1).

1. *Preliminaries.* The input tensor is denoted as $f(t, 1)$ for $t = 0, \ldots, N-1$. Construct an L level hierarchical bipartition of both domains if *Butterfly* coefficients are used as initial coefficients.

2. *Initialization (level $\ell = 0$).* 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 $f(:, 1)$ together with added bias term and ReLU activation. The weight coefficient tensor is denoted as $W_{k,q,1}^{(0)}$, where $k = 1, \ldots, r$ and $q = 1, \ldots, m$. This layer maps the input tensor f to an output tensor denoted as $\lambda^{(0)}(0, j, k)$, for $j = 0, \ldots, 2^L - 1$ being the index of data and $k = 1, \ldots, r$ being the index of channel. A general tensor notation $\lambda^{(\ell)}(i, j, k)$ corresponds to $\lambda_k^{A_i^\ell B_j^{L-\ell}}$ in *Butterfly* algorithm in Section 2.2.

According to the notations in (8), the weight coefficient tensor can be initialized as,

$$W^{(0)}_{k,q,1} \overset{\diamond}{=} e^{-2\pi\imath\xi_0\cdot(t_q-t_k)}\mathcal{L}_k(t_q), \quad 1 \leqslant k \leqslant r \text{ and } 1 \leqslant q \leqslant m, \tag{22}$$

where t_q and t_k denote uniform grid points and Chebyshev points in B_0^L respectively, and $\overset{\diamond}{=}$ denotes extended assign operator [1].

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.

3. *Recursion (level $\ell = 1, \ldots, L/2$).* The input tensor of layer ℓ is $\lambda^{(\ell-1)}(\lfloor i/2 \rfloor, 2j : 2j+1, s)$ for $i = 0, \ldots, 2^\ell - 1$ being the non-mixing channel index, $j = 0, \ldots, 2^{L-\ell} - 1$ being the data index and $s = 1, \ldots, r$ being the regular channel index. For each of the non-mixing channel i, one 1D convolution layer with filter size 2, stride 2 and output channel r is added together with bias term and ReLU activation. The weight coefficient tensors are denoted as and $W^{(\ell),i}_{k,c,s}$, where $k, s = 1, \ldots, r$ and $c = 0, 1$. Here c corresponds to the index of C being child of B. This layer maps the input tensor to output tensor, $\lambda^{(\ell)}(i, j, k)$.

   According to the notations in (11), the weight coefficient tensor can be initialized as,

$$W^{(\ell),i}_{k,c,s} \overset{\diamond}{=} e^{-2\pi\imath\xi_0^i\cdot(t_s^c-t_k)}\mathcal{L}_k(t_s^c) \tag{23}$$

   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 interval. Different frequency component in the input signal is now organized in different non-mixing channel. They will be transformed independently later which is related to the orthogonality of basis functions in different frequency intervals.

4. *Switch (level $\ell = L/2$).* This layer is a special layer of local operations. Denote the input tensor as $\lambda^{(L/2)}(i, j, s)$ and the dense coefficients as $W^{(L/2),i,j}_{k,s}$ for $i, j = 0, \ldots, 2^{L/2} - 1$ and $k, s = 1, \ldots, r$. For each i, j, $W^{(L/2)}$ is a r by r dense matrix. The operation at this level is as follows,

$$\lambda^{(L/2)}(i, j, k) = \sum_{s=1}^{r} W^{(L/2),i,j}_{k,s}\lambda^{(L/2)}(i, j, s) \tag{24}$$

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

   According to the notation in (13), the dense coefficients tensor can be initialized as,

$$W^{(L/2),i,j}_{k,s} \overset{\diamond}{=} e^{-2\pi\imath\xi_k^{A_i^{L/2}}\cdot t_s^{B_j^{L/2}}}, \tag{25}$$

   where $\xi_k^{A_i^{L/2}}$ and $t_s^{B_j^{L/2}}$ are Chebyshev points in $A_i^{L/2}$ and $B_j^{L/2}$ respectively.

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

5. *Recursion (level $\ell = L/2 + 1, \ldots, L$).* The input tensor of layer ℓ is $\lambda^{(\ell-1)}(\lfloor i/2 \rfloor, 2j : 2j+1, s)$ for $i = 0, \ldots, 2^\ell - 1$ being the data index, $j = 0, \ldots, 2^{L-\ell} - 1$ being the non-mixing channel index and $s = 1, \ldots, r$ being the regular channel index. The weight coefficient tensors are denoted as and $W^{(\ell),j}_{k,c,s}$,

[1] Since the coefficient is complex number and ReLU is activated, in order to reproduce the *Butterfly* algorithm in Tensorflow, which only handle real numbers, we have to extend the complex coefficients into four parts consistently, i.e., positive real, positive imaginary, negative real, negative imaginary. Hence $\overset{\diamond}{=}$ denotes the consistent assignment of complex coefficients. The same notation is used in the following content without further explanation.

where $k, s = 1, \ldots, r$ and $c = 0, 1$. Here c corresponds to the index of C being child of B. For each of the non-mixing channel j, one 1D convolution layer is performed as,

$$\lambda^{(\ell)}(i, j, k) = \sum_{s=1}^{r} \sum_{c=0,1} W_{k,c,s}^{(\ell),j,a} \lambda^{(\ell-1)}(\lfloor i/2 \rfloor, 2j + c, s). \tag{26}$$

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

According to the notations in (16), the weight coefficient tensor can be initialized as,

$$W_{k,c,s}^{(\ell),j,a} \stackrel{\diamond}{=} e^{-2\pi\imath(\xi_k^a - \xi_s)\cdot t_0^{2j+c}} \mathcal{L}_s(\xi_k^a) \tag{27}$$

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 complementary analog of step 3. Instead of organizing output in non-mixing frequency intervals, 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.

6. *Termination (level $\ell = L$).* Let $m = K/2^L$ 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 $\lambda^{(L)}(i, 0, s)$ together with added bias term and ReLU activation. The weight coefficient tensor is denoted as $W_{k,0,s}^{(L)}$, where $k = 1, \ldots, m$ and $s = 1, \ldots, r$. This layer maps the input tensor to an output tensor denoted as $\widehat{f}(i, k)$, for $i = 0, \ldots, 2^L - 1$ being the index of data and $k = 1, \ldots, m$ being the index of channel. Reshaping $\widehat{f}(p) = \widehat{f}(i, k)$ for $p = im + k$ gives a single vector output, which is analogy of the output vector of the *Butterfly* algorithm.

   According to the notations in (17), the weight coefficient tensor can be initialized as,

$$W_{k,0,s}^{(L)} \stackrel{\diamond}{=} e^{-2\pi\imath(\xi_k - \xi_s)\cdot t_0} \mathcal{L}_s(\xi_k), \quad 1 \leqslant k \leqslant m \text{ and } 1 \leqslant s \leqslant r, \tag{28}$$

   where ξ_k and ξ_s denote uniform grid points and Chebyshev points in A_0^L respectively, t_0 denotes the center of B_0^0.

   This step interpolates the transformed function from Chebyshev points back to uniform grid points.

7. *Task-dependent Layers.* Any type of layers, e.g., dense layer, convolution layer, transpose convolution layer, etc., can be built on top of $\widehat{f}$ and achieves 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*, Figure 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 Figure 2 (b), and the channel direction only contains non-mixing channels.

Butterfly-net can also be viewed as a specially constructed convolutional neural network. The initialization level and termination level are regular convolutional and transposed convolutional neural network. For recursion levels $\ell = 1, \ldots, L/2$, the non-mixing channel index i and regular channel index k can be combined and viewed as new channel index. *Butterfly-net* can be viewed as a convolutional neural network with sparsely connected channels under the new channel index. Similar observation applies to recursion levels $\ell = L/2 + 1, dots, L$, which can be viewed as a transposed convolutional neural network with sparsely connected channels if j-index and s-index are combined.

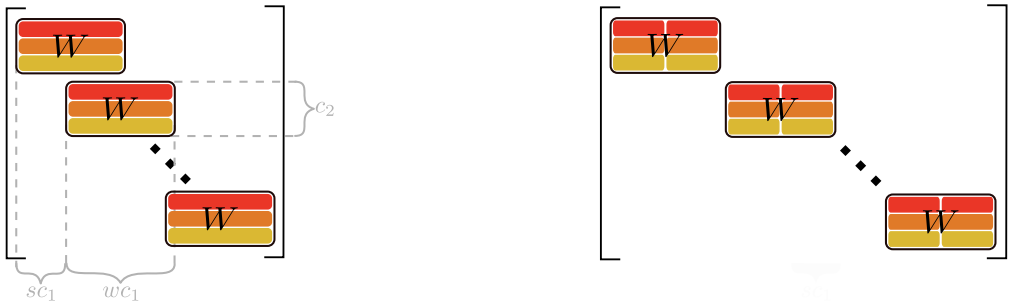
The main advantage of the proposed *Butterfly-net* is the reduction of model size and computational complexity. As can be seen from a simple calculation, the overall number of parameters involved is $O(r^2 N)$, where N is the size of input, and r is the size of mixing channels. Such a parametrization is near optimal for many well-known kernels, for instance, discrete Fourier kernel, since it gives almost linear scaling algorithms up to a logarithmic factor. And the overall computational cost for a evaluation of the *Butterfly-net* is $O(N \log N)$.

The extension of the *Butterfly-net* to d dimensional input signals, $f(t)$ for $t \in \mathbb{R}^d$, is straightforward. Under the same architecture, we can simply replace all indices i, j, k, s by their multi dimensional counterpart. This means instead of 1D mixing channels and 1D non-mixing channels, now we have dD mixing channels and dD non-mixing channels. The overall number of parameters in this case is then $O(r^{2d}N)$, where N is the total size of the d dimensional input. Another way of d dimensional extension can be achieved through the application of d one-dimensional *Butterfly-net* on each dimension of the input data. In the later numerical example, the later extension is adopted.

3.3 Matrix Representation of *Butterfly-net*

This section aims to demonstrate the matrix representation of *Butterfly-net*, which is similar to *Butterfly* factorization [30]. 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 an extra middle level local connection. Figure 3 (a) represents (19) 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. Figure 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.



(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

Since the CNN with mixing channel can already be represented as Figure 3, *Butterfly-net* simply stack interpolation convolutional layer together and drives 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. *Preliminaries.* No matrix representation is needed.
2. *Initialization (level $\ell = 0$).* The matrix representation is Figure 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.
3. *Recursion (level $\ell = 1, \ldots, L/2$).* For each $i = 0, \ldots, 2^\ell - 1$, the coefficient 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 $W^{(\ell),i}$. Then the convolutional operator of mixing and non-mixing channels can be viewed as the following matrices respectively,

$$H^{(\ell)}_{\lfloor i/2 \rfloor} = \begin{pmatrix} W^{(\ell),2\lfloor i/2 \rfloor} & & \\ & \ddots & \\ & & W^{(\ell),2\lfloor i/2 \rfloor} \\ \hline W^{(\ell),2\lfloor i/2 \rfloor+1} & & \\ & \ddots & \\ & & W^{(\ell),2\lfloor i/2 \rfloor+1} \end{pmatrix}, \quad \text{and } H^{(\ell)} = \begin{pmatrix} H^{(\ell)}_0 & & \\ & \ddots & \\ & & H^{(\ell)}_{2^{\ell-1}-1} \end{pmatrix}. \tag{29}$$

If we permute both input and output vector of current level, i.e., organize mixing and non-mixing channel together, then the matrix representation is permuted accordingly as,

$$\widetilde{H}_{CNN}^{(\ell)} = \left(\begin{array}{ccc} W^{(\ell),0} & & \\ & \ddots & \\ & & W^{(\ell),2^{\ell-1}-1} \\ \hline W^{(\ell),1} & & \\ & \ddots & \\ & & W^{(\ell),2^{\ell-1}} \end{array}\right), \quad \text{and } \widetilde{H}^{(\ell)} = \begin{pmatrix} \widetilde{H}_{CNN}^{(\ell)} & & \\ & \ddots & \\ & & \widetilde{H}_{CNN}^{(\ell)} \end{pmatrix}. \tag{30}$$

Hence the permuted operation can be viewed as a convolutional layer with sparse convolutional kernel $\widetilde{H}_{CNN}^{(\ell)}$ whereas regular CNN uses dense kernels.

4. *Switch (level $\ell = L/2$).* The matrix representation of switch layer is

$$M = \begin{pmatrix} M_{0,0} & \cdots & M_{0,2^{L/2}-1} \\ \vdots & \ddots & \vdots \\ M_{2^{L/2}-1,0} & \cdots & M_{2^{L/2}-1,2^{L/2}-1} \end{pmatrix}. \tag{31}$$

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

5. *Recursion (level $\ell = L/2+1, \ldots, L$).* For each $j = 0, \ldots, 2^{L-\ell}-1$ and $c = 0, 1$, the coefficient 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,

$$G_j^{(\ell)} = \left(\begin{array}{ccc|ccc} W_0^{(\ell),j} & & & W_1^{(\ell),j} & & \\ & \ddots & & & \ddots & \\ & & W_0^{(\ell),j} & & & W_1^{(\ell),j} \end{array}\right), \quad \text{and } G^{(\ell)} = \begin{pmatrix} G_0^{(\ell)} & & \\ & \ddots & \\ & & G_{2^{L-\ell}-1}^{(\ell)} \end{pmatrix}. \tag{32}$$

6. *Termination (level $\ell = L$).* The matrix representation is Figure 3 (b) with 2^L diagonal blocks and each block is $W_{:,0,:}^{(L)}$ which is of size $m \times r$. The resulting matrix is denoted as U.

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

$$g = \mathcal{B}(f) = UF\left[G^{(L)}F\left[\cdots G^{(L/2+1)}F\left[MF\left[H^{(L/2)}F\left[\cdots H^{(1)}F\left[Vf\right]\right]\right]\right]\right]\right], \tag{33}$$

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

4 Analysis of Approximation Power

The approximation power of *Butterfly-net* for Fourier kernel is first analyzed in Section 4.1, showing the same convergence rate as that of the original Butterfly algorithm, c.f. Theorem 4.1 in Section 4.1. Section 4.2 proves Theorem 4.1. Extension of the approximation analysis to task-dependent layers that follow Butterfly layers is discussed in Section 4.3.

4.1 Approximation of the Fourier Kernel

In this section, we analyze the approximation power of the *Butterfly-net* on a specific kernel, discrete Fourier kernel, whose matrix entry is defined as $\mathcal{K}_{ij} = e^{-2\pi\imath\xi_i \cdot t_j}$ where t_j and ξ_i are uniformly distributed on $[0, 1)$ and $[-\frac{K}{2}, \frac{K}{2})$ $(K \leqslant N)$ 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 data size, it exhibits full approximation power in terms of function representations.

Theorem 4.1. *Let N denote the size of the input and K denote the length of the domain of the output in the Butterfly-net. L and r are two parameters such that $\pi e K \leqslant r2^L$. 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 kernel such that for any bounded input vector f, the error of the output of the Butterfly-net satisfies that for any $p \in [1, \infty]$*

$$\|\mathcal{K}f - \mathcal{B}(f)\|_p \leqslant m^{1-\frac{1}{p}} C_{r,K} \left(\frac{\sqrt{r}\left(\frac{2}{\pi}\ln r + 1\right)}{2^{r-2}} \right)^L \|f\|_p, \tag{34}$$

where $m = N/2^L$, and $C_{r,K} = {(2+2/\pi \ln r)^3 (\pi e K)^r}/{(2r)^{r-2}}$ is a constant depending only on r and K.

The proof of Theorem 4.1 is constructive. We first fill the *Butterfly-net* with a specific set of parameters based on the complementary low-rank property of the discrete Fourier kernel (see Theorem 2.1). Once the parameters of *Butterfly-net* are constructed in this way, which means entries in the matrix representation of the *Butterfly-net* are explicitly known, 1-norm and ∞-norm of each matrix can be bounded. Combined with the low-rank approximation error at different levels, we derive the 1-norm and ∞-norm upper bound for the *Butterfly-net* matrix representation. Applying Riese-Thorin interpolation theorem, we reach to the conclusion of Theorem 4.1 for general norm index p. Section 4.2 provides the detailed proof of the theorem.

Previously, in the context of fast algorithms, Kunis and Melzer [27] analyzed the approximations of a simplified *Butterfly* scheme and Demanet et al. [14] 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 channel size in each layer, two parts of the error bound, the explicit power and the constant $C_{r,K}$, both decrease. The base of the power decays exponentially as the increase of r whereas the constant $C_{r,K}$ decays as r^{-r}. Interestingly, when L increases, which corresponds to increase the depth of the *Butterfly-net*, the error bound decays exponentially in L when $r > 3$ as can be verified by a simple calculation. Hence we conclude that the error of the *Butterfly-net* decays exponentially in L for any $r > 3$.

4.2 Proof of Theorem 4.1

This section first provides a few lemmas and their proof bounding each sparse matrix in (33). And then Theorem 4.1 is proved in detail.

Lemma 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, the Lebesgue constant Λ_r is bounded as*

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

Lemma 4.2 is a standard result of Chebyshev interpolation and the proof can be found in [42].

Corollary 4.3. *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 $i \leqslant r$,*

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

Corollary 4.3 is an immediate result of Lemma 4.2.

Lemma 4.4. *Let V be the block diagonal matrix defined in the preparation layer, then*

$$\|V\|_1 \leqslant \frac{2}{\pi}\ln r + 1 \text{ and } \|V\|_\infty \leqslant m\left(\frac{2}{\pi}\ln r + 1\right)$$

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

Proof. V is a block diagonal matrix with block V_j for $j = 1, 2, \ldots, 2^L$. V_j are the same $r \times m$ with entry $e^{-2\pi\imath\xi_0^{A_0^0}\cdot(t^{B_1^L} - t_k^{B_1^L})}\mathcal{L}_k(t^{B_j^L})$. By the definition of matrix 1-norm, we have

$$\|V\|_1 = \|V_1\|_1 \leqslant \max_{t\in B_1^L}\sum_{t_k^{B_1^L}}\left|e^{-2\pi\imath\xi_0^{A_0^0}\cdot(t-t_k^{B_1^L})}\mathcal{L}_k(t)\right| \leqslant \max_{t\in B_1^L}\sum_{t_k^{B_1^L}}|\mathcal{L}_k(t)| \leqslant \frac{2}{\pi}\ln r + 1. \tag{35}$$

By the definition of matrix ∞-norm, we have

$$\|V\|_\infty = \|V_1\|_\infty \leqslant \sum_{t\in B_1^L}\sum_{t_k^{B_1^L}}\left|e^{-2\pi\imath\xi_0^{A_0^0}\cdot(t-t_k^{B_1^L})}\mathcal{L}_k(t)\right| \leqslant m\left(\frac{2}{\pi}\ln r + 1\right). \tag{36}$$

□

Lemma 4.5. *Let M be the block diagonal matrix defined in the Layer M, then*

$$\|M\|_1 \leqslant r \text{ and } \|M\|_\infty \leqslant r,$$

where r is the number of Chebyshev points.

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

$$\|M\|_1 = \max_j\|M_j\|_1 \leqslant \max_j\max_i\|W_{j,i}\|_1 = \max_{j,i}\max_{t_k^{B_j^{L/2}}}\sum_{\xi_{k'}^{A_i^{L/2}}}\left|e^{-2\pi\imath\xi_{k'}^{A_i^{L/2}}\cdot t_k^{B_j^{L/2}}}\right| = r. \tag{37}$$

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

$$\|M\|_\infty = \max_j\|M_j\|_\infty \leqslant \max_j\max_i\|W_{j,i}\|_\infty = \max_{j,i}\max_{t_k^{B_j^{L/2}}}\sum_{t_k^{B_j^{L/2}}}\left|e^{-2\pi\imath\xi_{k'}^{A_i^{L/2}}\cdot t_k^{B_j^{L/2}}}\right| = r. \tag{38}$$

□

Lemma 4.6. *Let $H^{(\ell)}$ be the block diagonal matrix defined in the Layer $\ell = 1, \ldots, L/2$, then*

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

where r is the number of Chebyshev points.

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

$$\begin{aligned}
\left\|H^{\ell}\right\|_1 &= \max_i\left\|H_i^{(\ell)}\right\|_1 \leqslant \max_i\left(\left\|W_{2i-1}^{(\ell)}\right\|_1 + \left\|W_{2i}^{(\ell)}\right\|_1\right)\\
&\leqslant \max_i\left(\max_{t\in B_1^{L-\ell}}\sum_{t_k^{B_1^{L-\ell}}}\left|e^{-2\pi\imath\xi_0^{A_{2i-1}^{\ell}}\cdot(t-t_k^{B_1^{L-\ell}})}\mathcal{L}_k(t)\right| + \max_{t\in B_1^{L-\ell}}\sum_{t_k^{B_1^{L-\ell}}}\left|e^{-2\pi\imath\xi_0^{A_{2i}^{\ell}}\cdot(t-t_k^{B_1^{L-\ell}})}\mathcal{L}_k(t)\right|\right)\\
&\leqslant \max_i\left(\max_{t\in B_1^{L-\ell}}\sum_{t_k^{B_1^{L-\ell}}}|\mathcal{L}_k(t)| + \max_{t\in B_1^{L-\ell}}\sum_{t_k^{B_1^{L-\ell}}}|\mathcal{L}_k(t)|\right) \leqslant 2\left(\frac{2}{\pi}\ln r + 1\right).
\end{aligned} \tag{39}$$

□

Lemma 4.7. *Let $G^{(\ell)}$ be the block diagonal matrix defined in the Layer $\ell = L/2+1, \ldots, L$, then*

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

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

$$\begin{aligned}
\left\|G^{\ell}\right\|_1 =& \max_j \left\|G_j^{(\ell)}\right\|_1 \leqslant \max_j \left(\left\|W_{2j-1}^{(\ell)}\right\|_1 + \left\|W_{2j}^{(\ell)}\right\|_1\right) \\
\leqslant& \max_j \left\{ \max_k \left(\sum_{\xi_{k'}^{A_1^{\ell-1}}} \left| e^{-2\pi\imath(\xi_{k'}^{A_1^{\ell-1}} - \xi_k^{A_1^{\ell}} \cdot t_1^{B_{2j-1}^{L-\ell}})} \mathcal{L}_k(\xi_{k'}^{A_1^{\ell-1}}) \right| + \sum_{\xi_{k'}^{A_2^{\ell-1}}} \left| e^{-2\pi\imath(\xi_{k'}^{A_2^{\ell-1}} - \xi_k^{A_1^{\ell}} \cdot t_1^{B_{2j-1}^{L-\ell}})} \mathcal{L}_k(\xi_{k'}^{A_2^{\ell-1}}) \right| \right) \right. \\
& \left. + \max_k \left(\sum_{\xi_{k'}^{A_1^{\ell-1}}} \left| e^{-2\pi\imath(\xi_{k'}^{A_1^{\ell-1}} - \xi_k^{A_1^{\ell}} \cdot t_1^{B_{2j}^{L-\ell}})} \mathcal{L}_k(\xi_{k'}^{A_1^{\ell-1}}) \right| + \sum_{\xi_{k'}^{A_2^{\ell-1}}} \left| e^{-2\pi\imath(\xi_{k'}^{A_2^{\ell-1}} - \xi_k^{A_1^{\ell}} \cdot t_1^{B_{2j}^{L-\ell}})} \mathcal{L}_k(\xi_{k'}^{A_2^{\ell-1}}) \right| \right) \right\} \\
\leqslant& \max_j \left\{ \max_k \left(\sum_{\xi_{k'}^{A_1^{\ell-1}}} \left| \mathcal{L}_k(\xi_{k'}^{A_1^{\ell-1}}) \right| + \sum_{\xi_{k'}^{A_2^{\ell-1}}} \left| \mathcal{L}_k(\xi_{k'}^{A_2^{\ell-1}}) \right| \right) + \max_k \left(\sum_{\xi_{k'}^{A_1^{\ell-1}}} \left| \mathcal{L}_k(\xi_{k'}^{A_1^{\ell-1}}) \right| + \sum_{\xi_{k'}^{A_2^{\ell-1}}} \left| \mathcal{L}_k(\xi_{k'}^{A_2^{\ell-1}}) \right| \right) \right\} \\
\leqslant& 2r\left(\frac{2}{\pi}\ln r + 1\right).
\end{aligned} \tag{40}$$

□

Lemma 4.8. *Let $H^{(\ell)}$ be the block diagonal matrix defined in the Layer $\ell = 1, \ldots, L/2$, then*

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

where r is the number of Chebyshev points.

Lemma 4.9. *Let $G^{(\ell)}$ be the block diagonal matrix defined in the Layer $\ell = L/2+1, \ldots, L$, then*

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

where r is the number of Chebyshev points.

The proofs of Lemma 4.8 and Lemma 4.9 follow that of Lemma 4.7 and Lemma 4.6 respectively.

Proof of Theorem 4.1. Assume the *Butterfly-net* is full filled with the Forward *Butterfly-net* as in Section 2.2 with all rectifiers are deactivated. The kernel matrix is approximated by a product of sparse matrices, $U, G^{(\ell)}, M, H^{(\ell)}$, and V. We write the exact matrix product with error matrix $E^{(\ell)}$,

$$\begin{aligned}
\mathcal{K} = \Big[\Big[\Big[\Big[\Big[\Big[\Big[U + E^{(L)}\Big] G^{(L)} + E^{(L-1)}\Big] \cdots \Big] G^{(L/2+1)} + E^{(M)} \\
+ \quad M\Big] H^{(L/2)} + E^{(L/2)}\Big] \cdots \Big] H^{(1)} + E^{(1)}\Big] V + E^{(0)}.
\end{aligned} \tag{41}$$

In the following derivation, we adopt the notation, $\Lambda_r = \frac{2}{\pi}\ln r + 1$. Then, we have,

$$
\begin{aligned}
&\left\|\mathcal{K} - UG^{(L)}\cdots G^{(L/2+1)}MH^{(L/2)}\cdots H^{(1)}V\right\|_1 \\
\leqslant&\left\|E^{(L/2)}G^{(L)}\cdots G^{(L/2+1)}MH^{(L/2)}\cdots H^{(1)}V\right\|_1 \\
&+\cdots+\left\|E^{(M)}MH^{(L/2)}\cdots H^{(1)}V\right\|_1+\cdots+\left\|E^{(1)}V\right\|_1+\left\|E^{(0)}\right\|_1 \\
\leqslant&\left\|E^{(L/2)}\right\|_1\left(\prod_{\ell=L/2+1}^{L}\left\|G^{(\ell)}\right\|_1\right)\|M\|_1\left(\prod_{\ell=1}^{L/2}\left\|H^{(\ell)}\right\|_1\right)\|V\|_1 \\
&+\cdots+\left\|E^{(M)}\right\|_1\|M\|_1\left(\prod_{\ell=1}^{L/2}\left\|H^{(\ell)}\right\|_1\right)\|V\|_1+\cdots+\left\|E^{(1)}\right\|_1\|V\|_1+\left\|E^{(0)}\right\|_1 \\
\leqslant&\delta_1\sum_{\ell=0}^{L/2}\left(2r\Lambda_r\right)^{\ell}r\left(2\Lambda_r\right)^{L/2}\Lambda_r+\delta_1\sum_{\ell=0}^{L/2}\left(2\Lambda_r\right)^{\ell}\Lambda_r+\delta_1 \\
\leqslant&\left(\frac{\left(2r\Lambda_r\right)^{L/2+1}-1}{2r\Lambda_r-1}r\left(2\Lambda_r\right)^{L/2}\Lambda_r+\frac{\left(2\Lambda_r\right)^{L/2+1}-1}{2\Lambda_r-1}\Lambda_r+1\right)\delta_1 \\
\leqslant&\left(\left(2r\Lambda_r\right)^{L/2+1}\left(2\Lambda_r\right)^{L/2}+\left(2\Lambda_r\right)^{L/2+1}+1\right)\delta_1 \\
\leqslant&\left(4r\Lambda_r^2\right)^{L/2+1}\delta_1,
\end{aligned}
\tag{42}
$$

where the last few inequalities adopt the fact that $r > 1$ and $\Lambda_r > 1$, δ_1 is a uniform upper bound for $\left\|E^{(\ell)}\right\|_1$. Theorem 2.1 provides an upper bound for each entry of $E^{(\ell)}$. Hence, the uniform upper bound for the 1-norm is

$$
\delta_1 = \max_{\ell}\left\|E_1^{(\ell)}\right\|_1 \leqslant 2^L r\left(1+\Lambda_r\right)\left(\frac{2\pi eK}{4r2^L}\right)^r. \tag{43}
$$

Substituting the upper bound of δ_1 into (42), we obtain,

$$
\begin{aligned}
\left\|\mathcal{K} - UG^{(L)}\cdots G^{(L/2+1)}MH^{(L/2)}\cdots H^{(1)}V\right\|_1 \leqslant&\left(4r\Lambda_r^2\right)^{L/2+1}2^L r\left(1+\Lambda_r\right)\left(\frac{2\pi eK}{4r2^L}\right)^r \\
\leqslant&C_{r,K}\left(\frac{r\Lambda_r^2}{4^{r-2}}\right)^{L/2} = C_{r,K}\left(\frac{r\left(\frac{2}{\pi}\ln r+1\right)^2}{4^{r-2}}\right)^{L/2},
\end{aligned}
\tag{44}
$$

where $C_{r,K} = {4(2+2/\pi\ln r)^3 r^2(\pi eK)^r}/{(2r)^r}$ is a constant depends on r and K, and is independent of L. Equation (44) says that when r is chosen such that $\frac{r\Lambda_r^2}{4^{r-2}} < 1$ ($r > 3$ is sufficient), increasing L reduces the error exponentially. Here L is depth of the neural network.

On the other side, measuring the error under matrix ∞-norm, we have,

$$\begin{aligned}
&\left\|\mathcal{K}-UG^{(L)}\cdots G^{(L/2+1)}MH^{(L/2)}\cdots H^{(1)}V\right\|_\infty\\
\leqslant&\left\|E^{(L/2)}\right\|_\infty\left(\prod_{\ell=L/2+1}^{L}\left\|G^{(\ell)}\right\|_\infty\right)\|M\|_\infty\left(\prod_{\ell=1}^{L/2}\left\|H^{(\ell)}\right\|_\infty\right)\|V\|_1\\
&+\cdots+\left\|E^{(M)}\right\|_\infty\|M\|_\infty\left(\prod_{\ell=1}^{L/2}\left\|H^{(\ell)}\right\|_\infty\right)\|V\|_\infty+\cdots+\left\|E^{(1)}\right\|_\infty\|V\|_\infty+\left\|E^{(0)}\right\|_\infty\\
\leqslant&\delta_\infty\sum_{\ell=0}^{L/2}\left(2\Lambda_r\right)^\ell r\left(2r\Lambda_r\right)^{L/2}m\Lambda_r+\delta_\infty\sum_{\ell=0}^{L/2}\left(2r\Lambda_r\right)^\ell m\Lambda_r+\delta_\infty\\
\leqslant&\left(\frac{\left(2\Lambda_r\right)^{L/2+1}-1}{2\Lambda_r-1}r\left(2r\Lambda_r\right)^{L/2}m\Lambda_r+\frac{\left(2r\Lambda_r\right)^{L/2+1}-1}{2r\Lambda_r-1}m\Lambda_r+1\right)\delta_\infty\\
\leqslant&\left(m(2r\Lambda_r)^{L/2+1}\left(2\Lambda_r\right)^{L/2}+m(2r\Lambda_r)^{L/2+1}+1\right)\delta_\infty\\
\leqslant&(4r\Lambda_r^2)^{L/2+1}m\delta_\infty,
\end{aligned}\tag{45}$$

where δ_∞ is the uniform upper bound for $\left\|E^{(\ell)}\right\|_\infty$ satisfying

$$\delta_\infty=\max_\ell\left\|E_1^{(\ell)}\right\|_\infty\leqslant 2^L r\left(1+\Lambda_r\right)\left(\frac{2\pi eK}{4r2^L}\right)^r.\tag{46}$$

Therefore, we reach to the ∞-norm bound of the error,

$$\left\|\mathcal{K}-UG^{(L)}\cdots G^{(L/2+1)}MH^{(L/2)}\cdots H^{(1)}V\right\|_\infty\leqslant mC_{r,K}\left(\frac{r\Lambda_r^2}{4^{r-2}}\right)^{L/2}=mC_{r,K}\left(\frac{r\left(\frac{2}{\pi}\ln r+1\right)^2}{4^{r-2}}\right)^{L/2},\tag{47}$$

where $C_{r,K}$ is the same constant as in (44).

Applying Riesz-Thorin interpolation theorem together with (44) and (47), we obtain the error bound under the matrix p-norm,

$$\left\|\mathcal{K}-UG^{(L)}\cdots G^{(L/2+1)}MH^{(L/2)}\cdots H^{(1)}V\right\|_p\leqslant m^{1-\frac{1}{p}}C_{r,K}\left(\frac{r\left(\frac{2}{\pi}\ln r+1\right)^2}{4^{r-2}}\right)^{L/2},\tag{48}$$

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

Since the input vector f is bounded, we can also pick the shift parameters associated with every activation layer such that all of them are deactivated. Hence, we obtain, for any bounded vector f, the conclusion of Theorem 4.1, which is the direct result of (48). □

The result of Theorem 4.1 can be generalized by a careful investigation of the proof. First the domain of the discrete Fourier kernel, interval $[0,1)$ and $[-K/2,K/2)$, can be scaled and shifted. Theorem 4.1 holds as long as the product of the length of two intervals remain bounded by $\sqrt{NK}$. Further, through a parallel reading of the proof of Theorem 4.1 and [8, 29], we can show that similar theorem can be provided for smooth Fourier integral operators (FIOs), i.e., $e^{-2\pi\imath\Phi(\xi,t)}$ with smooth $\Phi(\xi,t)$ satisfying homogeneity condition of degree 1. The approximation to smooth FIOs enables the usages of *Butterfly-net* to represent a large class of elliptic operators and band limited operators, which we discuss more in the next subsection.

4.3 Approximation of energy functional and elliptic operators

Functionals $\varphi[f]$ of the input signal f that can be further expressed as functions of the Fourier coefficients of f, i.e.

$$\varphi[f]=\phi[\hat{f}],\tag{49}$$

where ϕ could be easier to approximate. Hence such functional $\varphi[f]$ could be efficiently approximated by the composition of a *Butterfly-net* (to approximate $\hat{f}$) and a shallow network (to approximate ϕ). Examples include but not limited to energy functionals (see Section 5.4) of band limited high frequency operators and elliptic operators, etc. In other words, *Butterfly-net* can be used as a module to extract Fourier representation of an input within a larger network, and followed by various task-dependent layers which can be fully-connected, convolutional, and so on.

The approximation of the whole network which represents $\varphi[f]$ can by analyzed by combining the analysis of *Butterfly-net* in Theorem 4.1 and that of the task-dependent layers which approximates ϕ. As a prototypical example, a large class of energy functional can be represented as

$$\phi[\hat{f}] = \sum_k c_k g(\hat{f}(k)), \tag{50}$$

where c_k are constants and $g(\cdot)$ is a known function, e.g. $g(z) = |z|^2$. To approximate such $\varphi[f]$, one can use a shallow network with fully connected layers to approximate ϕ, the accuracy which is guaranteed by the universal approximation result [1, 11, 24]. Specifically, suppose ϕ is approximated by the network mapping Φ, then these approximation results typically give a bound of $\|\phi - \Phi\|_\infty$ which converges to zero as network complexity grows, and the rate depending on the regularity of the function ϕ. Theorem 4.1 gives a bound of $\|B[f] - \hat{f}\|$, where B is the mapping of *Butterfly-net*. Assuming that ϕ has Lipschitz constant L_ϕ, which is satisfied in most applications, we then have that

$$\left|\varphi[f] - \Phi(B[f])\right| \leqslant \left|\phi(\hat{f}) - \phi(B[f])\right| + \left|\phi(B[f]) - \Phi(B[f])\right| \leqslant L_\phi \left\|\hat{f} - B[f]\right\| + \|\phi - \Phi\|_\infty, \tag{51}$$

which is controlled small by the two bounds.

The approximation of elliptic operators as well as many other functionals involving the Fourier components can be analyzed similarly. For example, Laplace operator may be represented as $\mathcal{K}^{-1}\mathcal{D}\mathcal{K}$, where $\mathcal{D}$ is a diagonal matrix, and $\mathcal{K}$ is a smooth FIO [13]. The extension of Theorem 4.1 gives error control of *Butterfly-net* approximation of the operator $\mathcal{K}$, and $\mathcal{D}$ may be approximated by a shallow network. Combining the error analysis gives approximation bound for Laplace operator by the whole network. In image and signal processing, Laplace operator is often used for diffusion related process, thus *Butterfly-net* provides an alternative to standard CNN in these applications with theoretically proved approximation power.

5 Numerical Results

We present four numerical experiments to demonstrate the approximation power with or without training of the *Butterfly-net*, comparison against CNN counterpart and its application to energy functional. The first numerical experiment shows that the approximation error without training of an initialized *Butterfly-net* decays exponentially as the increases of the network depth L, which verifies the conclusion of Theorem 4.1. Then we show that the training of *Butterfly-net* further refines the approximation error. In the third experiment, we construct and compare *Butterfly-net*s and their CNN counterparts with random and equivalent initialization. Also in this experiment, we test the transfer learning capability of both *Butterfly-net* and its CNN counterpart. In the last experiment, *Butterfly-net* and its CNN counterpart with a additional fully connect layer interlacing with ReLU layers are tested and compared in the approximation of the energy functional of 1D and 2D Poisson's equation. All implementations are in python 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, $r = 4$, which is sufficient large for the decay factor in Theorem 4.1 being smaller than one. The *Butterfly-net* is filled with the coefficients of *Butterfly* algorithm. The input vector in this example is of size $N = 16384$ whereas different sizes of the output vector are tested. The output vector represents integer frequency of the input function in the frequency domain $[-K/2, K/2)$.

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 the matrix representation of *Butterfly-net*. All bias terms and ReLu activations are disabled in this example.

Table 1: Approximation accuracy of the Fourier kernel by the *Butterfly-net*. Number of Chebyshev points is $r = 4$.

N	$K = 64$				$K = 256$			
	L	ϵ_1	ϵ_2	ϵ_∞	L	ϵ_1	ϵ_2	ϵ_∞
16384	6	5.0e−2	6.8e−2	5.7e−2	8	6.4e−2	8.9e−2	6.6e−2
16384	8	1.9e−4	3.0e−4	2.4e−4	10	2.4e−4	3.8e−4	2.7e−4
16384	10	1.2e−6	1.3e−6	1.0e−6	12	8.6e−7	1.5e−6	1.2e−6

Table 1 shows for both choices of K, the relative approximation errors measured in matrix 1-norm, 2-norm, and ∞-norm decay exponentially as L increases. 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 4.1.

5.2 Approximation Power After Training

The second numerical experiment in this section aims to demonstrate the approximation power after training of *Butterfly-net*. Both the training and testing data are generated as follows. We first generate an array of $N/2$ random number. The first entry of the array (indexed by zero) is a uniform random real number in $[-2, 2)$ whereas the following $N/2 - 1$ numbers are uniform random complex number with both real and complex part being uniform in $[-1, 1)$. Second, we multiply the array entry-wise by a Gaussian function of index with mean G_{mean} and standard deviation G_{std}. The array then is complex symmetrized to be the frequency vector and an inverse discrete Fourier transform is applied to obtain the input function. Due to the special construction of zero frequency and complex symmetrization, the input function is a real signal. The parameters of the Gaussian mask determines the frequency window of the input signal. In this section, we test two families of input signals: **low-freq input** adopts $G_{\text{mean}} = 0$ and $G_{\text{std}} = 10$ whereas **high-freq input** adopts $G_{\text{mean}} = 40$ and $G_{\text{std}} = 10$.

Given these data generation procedure, all *Butterfly-net* with different initializations and frequency windows are trained in the infinity data setting, i.e., training data is randomly generated on the fly. Batch size is 256 and ADAM optimizer is used with learning rate being 0.001. The relative error is reported on $100,000$ test data in vector two norm. Default values are used for other unspecified hyper parameters.

	Initialization	Pre-Train Loss	Train Loss	Test Loss	Test Rel Err
low-freq input	prefix	2.81e−4	3.04e−6	3.04e−6	5.14e−2
	random	1.28e+1	6.92e−4	6.90e−4	7.72e−1
high-freq input	prefix	1.02e−4	3.31e−6	3.36e−6	7.81e−2
	random	4.48e−1	5.44e−4	5.51e−4	1.00e+0

Table 2: Parameters and training results of *Butterfly-net*. *Butterfly-net* uses 8 mixing channels and 6 layers. The frequency window is set to be $[0, 64)$.

Table 2 shows the training and testing results for *Butterfly-net*. For prefixed initial parameters, *Butterfly-net* is able to achieve good accuracy already without training and training further adjust the parameters to be input data distribution dependent and achieves two more digits accuracy in loss. Random initialization does not provide any accuracy in the beginning. Even after training, they are not able to achieve the loss as prefixed counterparts without training. All losses and relative error for randomly initialized *Butterfly-net*s are significantly worse than their prefixed counterparts. Hence for the *Butterfly-net* we constructed, the near-optimal coefficients are difficult to be discovered by stochastic gradient descent methods. Such an observation also carries over to the CNN counterpart of *Butterfly-net*. Another important feature in the

results is that *Butterfly-net* achieves similar accuracy for both low frequency and high frequency input data. While in the usual CNN literature, it is commonly believed that mappings of low frequency data are easier to be learned by CNNs.

5.3 Comparison to CNN and Transfer Learning

This numerical experiment is to compare the *Butterfly-net* and its CNN counterpart and test their transfer learning capability. Both the training and testing data are generated similar as above. We first generate an array of $N/2$ random number. The first entry of the array (indexed by zero) is a standard Gaussian random number whereas the following $N/2 - 1$ numbers are random complex number with both real and complex part being standard Gaussian random. Second, we multiply the array entry-wise by a Gaussian function of index with mean G_{mean} and standard deviation $G_{\text{std}} = 2$. The array then is complex symmetrized to be the frequency vector and an inverse discrete Fourier transform is applied to obtain the input function. The input function is again a real signal. In this section, we train two neural networks with low frequency data, i.e., $G_{\text{mean}} = 0$ and test them on data generated from different Gaussian masks, i.e., $G_{\text{mean}} = 0, 0.2, 0.4, \ldots, 5$.

	Initialization	# Parameters	Train Loss	Test Loss	Test Rel Err
Butterfly-net	*Butterfly* coefs	8584	7.30e−5	1.29e−4	3.12e−2
CNN	*Butterfly* coefs	15752	1.45e−4	1.37e−4	2.96e−2
CNN	Random	15752	1.72e−1	1.80e−1	1.00e+0

Table 3: Parameters and training results of *Butterfly-net* and its CNN counterpart with different initializations. All neural networks use 8 mixing channels and 6 layers. The frequency window is set to be $[0, 128)$.

The first two neural networks are initialized with coefficients in *Butterfly* algorithm and the last CNN network is initialized randomly. All of them are trained with infinity data setting and batch size being 256. ADAM optimizer is used with learning rate being 0.001 for *Butterfly-net* and 0.0005 for CNN counterpart. The relative errors are reported on $100,000$ test data in vector two norm. Given a fixed Gaussian mask, the same test data is used for both neural networks. Default values are used for other unspecified hyper parameters.

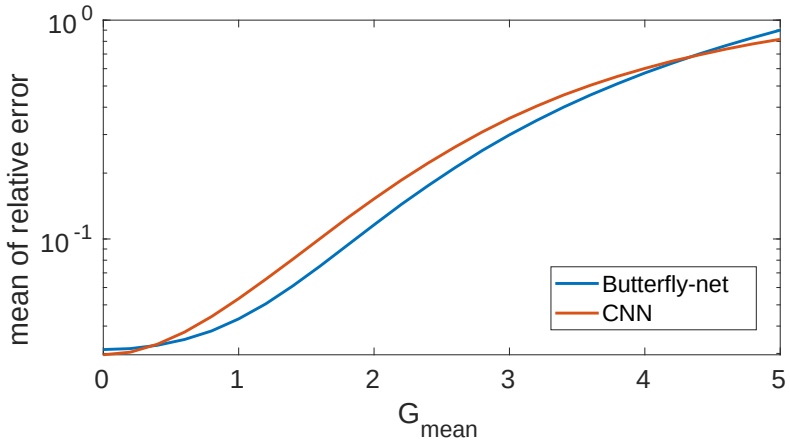


Figure 4: Mean of relative test error the *Butterfly-net* and CNN on test data with different G_{mean}.

Table 3 lists all results of all three neural networks without transferring. Clearly, we observe that with proper initialization, *Butterfly-net* achieves comparable accuracy as CNN while the number of parameters is about 2 times smaller. Less parameters in part means that *Butterfly-net* is less adaptive to the data distribution would have better generalizability. However, for CNN with random initialization, the network fails in approximating the output. More aggressive over parametrization of CNN eventually will approximate

the output due to the universal approximation theory and stochastic gradient methods. According to our results, mild over parametrization is not able to recover the parameters in either of the other two networks.

Figure 4 shows the results of generalizability comparison of *Butterfly-net* and its CNN counterpart, which validates the statement. The mean of relative transferred test error for both networks are illustrated. We observe that the red curve of *Butterfly-net* is almost always above that of CNN except the near $G_{\text{mean}} = 0$ and $G_{\text{mean}} = 5$. CNN achieves better accuracy near $G_{\text{mean}} = 0$ is expected. This is because on the training dataset or dataset sufficiently close CNN has 2 times more parameters and could achieve slightly better relative error. On the other hand, when $G_{\text{mean}} = 5$, both networks fail to provide reasonable prediction of the output. The comparison in this case is not informative. *Butterfly-net* achieves better test relative error for all other transferred test datasets. Hence we conclude that *Butterfly-net* has better generalizability than that of CNN.

5.4 Energy of Discrete Laplace Operators

The last numerical example aims to construct an approximation of the energy functional of 1D and 2D Poisson's equations, as explained in the Section 4 about the general function representation power of the *Butterfly-net*. Similar as above, we use a *Butterfly-net* with extra task-dependent layer and its CNN counterpart. We aims to obtain the energy of Poisson's equation $\Delta u = f$ with periodic boundary condition, where Δ denotes the Laplace operator, f is the input function, and u is the solution function. The energy functional of Poisson's equation is defined as the negative inner product of u and f, which can also be approximated by a quadratic form of the leading low-frequency Fourier components, which can be rewritten as,

$$\mathcal{E}(f) = -\langle f, u\rangle \approx \sum_{k\in[-K/2,K/2-1)^d} \frac{1}{|k|^2}\left|\widehat{f}_k\right|^2, \tag{52}$$

where $\widehat{f}_k$ is the k-th Fourier component of f and d is the dimension. If the input function f is a linear combination of the Fourier components with $k \in [-K/2, K/2)^d$, equality is achieved in (52).

In this numerical example, we assume the domain of f, $[0,1)^d$, is discredited by a uniform grid with 4096 points in 1D and 256 points on each dimension in 2D. f is a smooth periodic random function. It is generated from the Fourier interpolation of a fully random function defined on 512 1D grid and 32×32 2D grid. The reference energy is calculated via the discrete Laplace operator. Infinity data setting is used in training whereas 10^5 and 10^3 random instances are used as testing data in 1D and 2D respectively. In 1D example, *Butterfly-net* with 16 mixing channels are used and outputs 128 1D frequency components and 32×32 2D frequency components and CNN counterpart adopts the same configurations except all channels are mixed. In 2D example, we trained two *Butterfly-net*s and two CNNs with same configurations for each dimension and apply them as tensor product of operators. Two types of task-dependent layers are tested. The first one is square layer which is analog of (52) with coefficient being parameters. And the second type adopts dense layer with 128 neurons. For all examples, the batch size is 256. Networks are initialized with *Butterfly* coefficients and trained by ADAM with stepsize being 0.001 for square task-dependent layer cases and 0.01 for dense task-dependent layer cases. Default values are used for other unspecified parameters.

Task Layer	Network	# Parameters		Relative Error	
		Base Layers	Task Layer	Mean	Std
Square Layer	*Butterfly-net*	33296	256	3.02e−3	2.48e−3
	CNN	61968	256	3.13e−3	2.58e−3
128 Dense Layer	*Butterfly-net*	33296	33024	1.36e−2	1.15e−2
	CNN	61968	33024	1.11e−2	9.65e−3

Table 4: Parameters and training results of *Butterfly-net* and its CNN counterpart for 1D example. Layers in both networks are initialized with *Butterfly* coefficients. Both neural networks use 16 mixing channels and 6 layers.

Task Layer	Network	# Parameters		Relative Error	
		Base Layers	Task Layer	Mean	Std
Square Layer	*Butterfly-net*	22048	4096	9.09e−3	6.53e−3
	CNN	30240	4096	1.03e−2	7.60e−3
128 Dense Layer	*Butterfly-net*	22048	524544	1.20e−2	9.11e−3
	CNN	30240	524544	1.02e−2	7.66e−3

Table 5: Parameters and training results of *Butterfly-net* and its CNN counterpart for 2D example. Layers in both networks are initialized with *Butterfly* coefficients. Both neural networks use 16 mixing channels and 4 layers.

Table 4 and Table 5 show the results for energy of 1D and 2D Laplace operators. For both 1D and 2D case, we achieve similar conclusions. First of all, the number of *Butterfly-net*'s parameters in base layers is smaller than that of CNN counterpart, while similar or even better accuracies are achieved for all comparisons. Different task layers are tested in both cases, i.e., square layer and dense layer 128 width. For (52), it is known that square layer is the layer to use as long as the base layers approximates the Fourier transform. Hence a precise theoretical error bound is achievable for *Butterfly-net* and CNN counterpart with square layer. According to Table 4 and Table 5, square layer achieves better accuracy than dense layers. And *Butterfly-net* in both cases outperforms CNN. Based on the universal approximation theorem [1, 11, 24], dense layer is able to approximate arbitrary function if the width is wide enough. Here we adopts 128 being the width for comparison against square layer. Dense layer with is not as accurate as that of square layers, while the number of parameters for task layer is significantly increased. Hence, if a known close form of the mapping is available, it is suggested to adopt the close form as much as possible such that the number of parameter is minimized and better accuracy is achievable.

6 Conclusion and Discussion

A low-complexity CNN with structured *Butterfly* coefficients and sparse across-channel connections is proposed, motivated by the *Butterfly* scheme. The hierarchical functional representation by *Butterfly-net* is optimal in the sense that the model complexity is $O(N)$ and the computational complexity is $O(N \log N)$. The approximation accuracy to the Fourier kernel is proved to exponentially converge as the depths of the *Butterfly-net* increases, which provides an approximation upper bound for CNNs in a large class of problems in scientific computing and image and signal processing.

The trained *Butterfly-net*s from *Butterfly* coefficient initialization and random initialization are applied to represent discrete Fourier transforms and energy functionals of 1D and 2D Laplace equations. For these examples, *Butterfly-net* achieves better accuracy than its no-trained version. We also compared *Butterfly-net* against its CNN counterpart. From the numerical results, we find that *Butterfly-net* is able to achieve similar accuracy as CNN, while the number of parameters is orders of magnitudes smaller. In the transfer learning settings, *Butterfly-net* generalizes better than CNN 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 and ACI-1450280. XC is partially supported by the National Science Foundation via grants DMS-1818945 and DMS-1820827.

References

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

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

[3] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. Deep learning. *Nature*, 521(7553):436–444, 2015.

[4] Jens Berg and Kaj 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.

[5] 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.

[6] Jian-Feng Cai, Bin Dong, Stanley Osher, and Zuowei Shen. Image restoration: total variation, wavelet frames, and beyond. *Journal of the American Mathematical Society*, 25(4):1033–1089, 2012.

[7] Emmanuel J. Candès, Laurent Demanet, and Lexing 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.

[8] Emmanuel J. Candès, Laurent Demanet, and Lexing 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.

[9] Tony F Chan and Jianhong Jackie Shen. *Image processing and analysis: variational, PDE, wavelet, and stochastic methods*, volume 94. SIAM, 2005.

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

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

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

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

[14] Laurent Demanet, Matthew Ferrara, Nicholas Maxwell, Jack Poulson, and Lexing 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.

[15] Weinan E, Jiequn Han, and Arnulf 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.

[16] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

[17] Yuwei Fan and Lexing Ying. Solving electrical impedance tomography with deep learning, jun 2019. http://arxiv.org/abs/1906.03944.

[18] Yuwei Fan, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núñez. A multiscale neural network based on hierarchical matrices, 2018. https://arxiv.org/abs/1807.01883.

[19] Yuwei Fan, Jordi Feliu-Fabà, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núñez. A multiscale neural network based on hierarchical nested bases. *Res. Math. Sci.*, 6(2):21, mar 2019. doi: 10.1007/s40687-019-0183-3.

[20] Yuwei Fan, Cindy Orozco Bohorquez, and Lexing Ying. BCR-Net: A neural network based on the nonstandard wavelet form. *J. Comput. Phys.*, 384:1–15, may 2019. doi: 10.1016/J.JCP.2019.02.002.

[21] Jordi Feliu-Faba, Yuwei Fan, and Lexing Ying. Meta-learning pseudo-differential operators with deep neural networks, jun 2019. http://arxiv.org/abs/1906.06782.

[22] A Ronald Gallant and Halbert 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.

[23] Juncai He and Jinchao 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`.

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

[25] Yuehaw Khoo and Lexing Ying. SwitchNet: a neural network model for forward and inverse scattering problems, oct 2018. http://arxiv.org/abs/1810.09675.

[26] Yuehaw Khoo, Jianfeng Lu, and Lexing 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`.

[27] Stefan Kunis and Ines 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`.

[28] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.

[29] Yingzhou Li and Haizhao Yang. Interpolative butterfly factorization. *SIAM J. Sci. Comput.*, 39(2): A503–A531, 2017. URL `http://dx.doi.org/10.1137/16M1074941`.

[30] Yingzhou Li, Haizhao Yang, Eileen R. Martin, Kenneth L. Ho, and Lexing Ying. Butterfly factorization. *Multiscale Model. Simul.*, 13(2):714–732, jan 2015. URL `http://epubs.siam.org/doi/10.1137/15M1007173`.

[31] Yingzhou Li, Haizhao Yang, and Lexing Ying. A multiscale butterfly algorithm for multidimensional Fourier integral operators. *Multiscale Model. Simul.*, 13(2):1–18, jan 2015. URL `http://epubs.siam.org/doi/10.1137/140997658http://arxiv.org/abs/1411.7418`.

[32] Yingzhou Li, Haizhao Yang, and Lexing 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`.

[33] Yingzhou Li, Jianfeng Lu, and Anqi Mao. Variational training of neural network approximations of solution maps for physical models, may 2019. http://arxiv.org/abs/1905.02789.

[34] Shiyu Liang and R Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

[35] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-net: Learning PDEs from data. 80: 3208–3216, 2018. URL `http://proceedings.mlr.press/v80/long18a.html`.

[36] Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.

[37] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. Learning functions: when is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.

[38] Hrushikesh N Mhaskar and Tomaso Poggio. Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, 14(06):829–848, 2016.

[39] Eric Michielssen and Amir 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/`.

[40] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[41] Michael O'Neil, Franco Woolfe, and Vladimir Rokhlin. An algorithm for the rapid evaluation of special function transforms. *Appl. Comput. Harmon. Anal.*, 28(2):203–226, 2010.

[42] Theodore J. Rivlin. *Chebyshev polynomials: from approximation theory to algebra and number theory*. Wiley-Interscience, 2nd edition, 1990.

[43] Elia Schneider, Luke Dai, Robert Q Topper, Christof Drechsel-Grau, and Mark E Tuckerman. Stochastic neural network approach for learning high-dimensional free energy surfaces. *Physical review letters*, 119 (15):150601, 2017.

[44] Matus Telgarsky. Benefits of depth in neural networks. 49:1517–1539, 2016. URL `http://proceedings.mlr.press/v49/telgarsky16.html`.

[45] Dmitry Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94: 103–114, 2017.

[46] Lexing 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`.

[47] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan 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`.

[48] Ding-Xuan 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`.

A Proof of Theorem 2.1

Here we first include a well-known lemma of Chebyshev interpolation, Lemma A.1 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,*

$$\|f - \Pi_r f\|_\infty \leqslant \left(2 + \frac{2}{\pi} \ln r\right) \inf_{g \in \mathcal{P}_r} \|f - g\|_\infty. \tag{53}$$

The proof of Lemma A.1 can be found in [42].

Proof of Theorem 2.1. At level ℓ, we have $w(A^\ell) = {}^{K}\!/_{2^\ell}$ and $w(B^{L-\ell}) = {}^{1}\!/_{2^{L-\ell}}$, which implies $w(A^\ell) \cdot w(B^{L-\ell}) = {}^{K}\!/_{2^\ell 2^{L-\ell}} = K \cdot 2^{-L}$, where $w(\cdot)$ denote the function of length.

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

$$\begin{aligned}\mathcal{K}(\xi,t) =& e^{-2\pi\imath(\xi\cdot t-\xi_0\cdot t-\xi\cdot t_0+\xi_0\cdot t_0)}\cdot e^{-2\pi\imath\xi_0\cdot t}\cdot e^{-2\pi\imath\xi\cdot t_0}\cdot e^{2\pi\imath\xi_0\cdot t_0}\\ =& e^{-2\pi\imath R(\xi,t)}\cdot e^{-2\pi\imath\xi_0\cdot t}\cdot e^{-2\pi\imath\xi\cdot t_0}\cdot e^{2\pi\imath\xi_0\cdot t_0},\end{aligned} \tag{54}$$

where $R(\xi, t) = (\xi - \xi_0) \cdot (t - t_0)$.

Next, we show the r-term truncation error of the first term in the second line of (54). Based on the power expansion of $e^{-2\pi\imath R(\xi,t)}$, i.e.,

$$e^{-2\pi\imath R(\xi,t)} = \sum_{k=0}^{\infty} \frac{(-2\pi\imath R(\xi,t))^k}{k!}, \tag{55}$$

the r-term truncation error can be bounded as,

$$\begin{aligned}\delta =& \left| e^{-2\pi\imath R(\xi,t)} - \sum_{k=0}^{r} \frac{(-2\pi\imath R(\xi,t))^k)}{k!} \right| = \left| \sum_{k=r+1}^{\infty} \frac{(-2\pi\imath R(\xi,t))^k}{k!} \right| \\ \leqslant& \sum_{k=r+1}^{\infty} \frac{1}{k!} \left(\frac{2\pi K}{4\cdot 2^L}\right)^k \leqslant \sum_{k=r+1}^{\infty} \frac{e^k}{k^k} \left(\frac{2\pi K}{4\cdot 2^L}\right)^k \leqslant \sum_{k=r+1}^{\infty} \left(\frac{2\pi e K}{4r2^L}\right)^k \leqslant \left(\frac{2\pi e K}{4r2^L}\right)^r,\end{aligned} \tag{56}$$

where the last inequality uses $\pi e K \leqslant r2^L$. 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,

$$\left\| e^{-2\pi\imath R(\xi,t)} - \sum_{k=1}^{r} e^{-2\pi\imath R(\xi,t_k)} \mathcal{L}_k(t) \right\| \leqslant \left(2 + \frac{2}{\pi}\ln r\right)\delta. \tag{57}$$

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

$$\left\| e^{-2\pi\imath\xi\cdot t} - \sum_{k=1}^{r} e^{-2\pi\imath\xi\cdot t_k} e^{-2\pi\imath\xi_0\cdot(t-t_k)} \mathcal{L}_k(t) \right\| \leqslant \left(2 + \frac{2}{\pi}\ln r\right)\left(\frac{2\pi e K}{4r2^L}\right)^r, \tag{58}$$

for any $\xi \in A^\ell$ and $t \in B^{L-\ell}$.

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. □