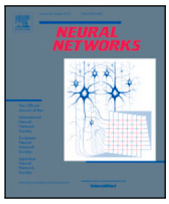
Neural Networks 187 (2025) 107277

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

Neural networks trained by weight permutation are universal approximators

Yongqiang Cai [a], Gaohang Chen [b,*], Zhonghua Qiao [c]

[a] School of Mathematical Sciences, Laboratory of Mathematics and Complex Systems, MOE, Beijing Normal University, Beijing, 100875, China
[b] Department of Applied Mathematics, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
[c] Department of Applied Mathematics & Research Institute for Smart Energy, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

ARTICLE INFO

Dataset link: https://github.com/DanclaChen/PermutationTraining

MSC:
41A30
68T05
68T07

Keywords:
Universal approximation property
Neural networks
Training algorithm
Learning behavior

ABSTRACT

The universal approximation property is fundamental to the success of neural networks, and has traditionally been achieved by training networks without any constraints on their parameters. However, recent experimental research proposed a novel permutation-based training method, which exhibited a desired classification performance without modifying the exact weight values. In this paper, we provide a theoretical guarantee of this permutation training method by proving its ability to guide a ReLU network to approximate one-dimensional continuous functions. Our numerical results further validate this method's efficiency in regression tasks with various initializations. The notable observations during weight permutation suggest that permutation training can provide an innovative tool for describing network learning behavior.

1. Introduction

The Universal Approximation Property (UAP) of neural networks is a cornerstone in the theoretical guarantee of deep learning, proving that even the simplest two-layer feedforward networks can approximate any continuous function (Cybenko, 1989; Hornik, Stinchcombe, & White, 1989; Leshno, Lin, Pinkus, & Schocken, 1993). This fascinating ability allows neural networks to replace critical, challenging components in existing frameworks to enhance efficiency (Lu, Jin, Pang, Zhang, & Karniadakis, 2021; Raissi, Perdikaris, & Karniadakis, 2019). Despite the extensive study in various settings (Castro, Mantas, & Benıtez, 2000; Fan, Xiong, & Wang, 2020; Wang & Qu, 2022), existing UAP research rarely imposes restrictions on the network parameters. However, in specific application cases, constraints are essential to meet certain requirements (Kosuge, Hamada, & Kuroda, 2021; Nugent, 2005).

As a constrained scenario, Qiu and Suda (2020) empirically showed that, without altering the exact value of network weights, only permuting the initialized weights can achieve comparable or better performance for image classification tasks. This unique property makes the permutation-based training method attractive for specific hardware applications, and has been utilized as fixed-weight accelerators (Kosuge, Hamada, & Kuroda, 2021). It can also facilitate the implementation of physical neural networks (Feldmann et al., 2021; Nugent, 2005). Despite its impressive benefits in applications, the permutation training method suffers from an absence of theoretical guarantees regarding its effectiveness, which hinders further algorithmic and hardware development necessary to fully exploit its potential.

This paper establishes the first theoretical foundation of this method (to our best knowledge) by proving the UAP of a permutation-trained Rectified Linear Unit (ReLU) network with random initializations for any one-dimensional continuous function. Compared to the conventional UAP scenarios, the proof of permutation training encounters a significantly greater challenge, primarily due to the extreme constraints of maintaining the initialized weight values. The key proof idea is a four-pair construction of the step function approximators, which enables the approximation through a piecewise constant function (Stein & Shakarchi, 2009). Additionally, a reorganization method is proposed to eliminate the impact of the remaining weights.

Our numerical experiments not only validate our theoretical results by illustrating the widespread existence of the UAP of permutation training in diverse initializations, but also emphasize the effects of initializations on the permutation training performance. Moreover, the patterns observed during permutation training also highlight its potential in describing learning behavior, relating to topics like the pruning technique (Frankle & Carbin, 2019) and continual learning (Maltoni & Lomonaco, 2019; Zeng, Chen, Cui, & Yu, 2019). Our main findings are summarized below:

* Corresponding author.
E-mail addresses: caiyq.math@bnu.edu.cn (Y. Cai), gaohang.chen@connect.polyu.hk (G. Chen), zqiao@polyu.edu.hk (Z. Qiao).

https://doi.org/10.1016/j.neunet.2025.107277
Received 15 June 2024; Received in revised form 11 November 2024; Accepted 12 February 2025
Available online 6 March 2025

1. We prove the UAP of permutation-trained ReLU networks with pairwise random initialization to one-dimensional continuous functions.
2. The numerical experiments of regression problems emphasize the crucial role played by the initializations in the permutation training scenario.
3. By observing the permutation patterns, we find that permutation training as a new approach holds promise in describing intricate learning behaviors.

1.1. Permutation training's advantages in hardware implementation

Permutation training has been applied to training large models on extensive databases for image classification tasks (Qiu & Suda, 2020), achieving performance that is comparable or even superior to conventional free-training training methods like Adam (Kingma & Ba, 2015). However, there is currently no evidence to report a significant advantage when applied to more diverse tasks on contemporary Graphics Processing Unit (GPU)-based hardware. Nevertheless, we believe it is highly suitable for the design of *physical neural networks* (Nugent, 2005), since reconnecting the neurons is sometimes more convenient than altering the exact value. Therefore, permutation training may inspire alternative physical weight connection implementations, such as using fixed-weight devices controlled by a permutation circuit (Qiu & Suda, 2020). This idea has been applied to a fixed-weight network accelerator (Kosuge, Hamada, & Kuroda, 2021; Kosuge, Hsu, Hamada, & Kuroda, 2021).

Another potential application scenario is the physical neural networks with an explicit structure to store the weight value, such as the *integrated photonic tensor core*, a computing chip with specialized architecture (Feldmann et al., 2021). This design has been successfully employed by international commercial companies in their photonic computing products. In each photonic tensor core, an array of phase-change cells are organized to separately store each element of the weight matrix, with their values adjusted through optical modulation of the transmission states of the cells. However, permutation training indicates that, in addition to changing the exact value, it is feasible to connect each cell with the permutation circuit for convenient reconnections. Therefore permutation training can facilitate the learning process.

1.2. Related works

The UAP has been extensively studied in various settings, leading to many efficient applications. It is well known that fully connected networks are universal approximators for continuous functions (Cybenko, 1989; Hornik et al., 1989; Leshno et al., 1993). Further research primarily concentrated on how the width and depth of models influence their UAP. An estimation has been made of the lower bound for the minimum width required by networks to achieve UAP (Cai, 2023; Lu, Pu, Wang, Hu, & Wang, 2017). It has also been demonstrated that the depth of the network is crucial, as increasing the network depth can enhance the expression power (Shen, Yang, & Zhang, 2022; Telgarsky, 2016; Yarotsky, 2017). Our work considers the network with arbitrary width, aligning with conventional scenarios. However, the constraints on parameters imposed by permutation training introduce non-trivial complexities and challenges not encountered in traditional free training methods.

Permutation, as a typical group structure, has been systematically described (Cameron, 1999). In deep learning, it closely relates to permutation equivariant or invariant networks (Cohen & Welling, 2016) designed to learn from symmetrical data (Lee et al., 2019; Zaheer et al., 2017). It is also evident in graph-structured data which inherently exhibit permutation invariance (Maron, Ben-Hamu, Shamir, & Lipman, 2019; Satorras, Hoogeboom, & Welling, 2021). However, permutation training is not limited to the issues with intrinsic symmetry.

As for the weight permutation attempts, Qiu and Suda (2020) empirically proposed the first (to our knowledge) weight-permuted training method. This method preserves the initialized weight value, allowing more efficient and reconfigurable implementation of the physical neural networks (Kosuge, Hamada, & Kuroda, 2021; Kosuge, Hsu, et al., 2021). Our work provides theoretical guarantees of this method and considers some regression tasks numerically. Additionally, initialization can be improved by rewiring neurons from the perspective of computer networks (Scabini, De Baets, & Bruno, 2024), but the training methods are unchanged.

Permutation training is also closely related to the permutation symmetry and Linear Mode Connectivity (LMC) (Entezari, Sedghi, Saukh, & Neyshabur, 2021; Frankle, Dziugaite, Roy, & Carbin, 2020). The LMC suggests that after a proper permutation, most stochastic gradient descent solutions under different initialization will fall in the same basin in the loss landscape. Similarly, permutation training also seeks a permutation to improve performance. Therefore, the search algorithms utilized in LMC indicate the possibility of more efficient permutation training algorithms, as the fastest algorithm can search a proper permutation of large ResNet models in seconds to minutes (Ainsworth, Hayase, & Srinivasa, 2023; Jordan, Sedghi, Saukh, Entezari, & Neyshabur, 2023).

1.3. Outline

We state the main result and proof idea in Section 2. In Section 3, we provide a detailed construction of the proof. The numerical results of permutation training are presented in Section 4, along with the observation of permutation behavior during the training process. In Section 5 we discuss the possible future work. Finally, the conclusion is provided in Section 6.

2. Notations and main results

This section introduces the notations and UAP theorems, accompanied by a brief discussion of the proof idea.

2.1. Neural networks architecture

We start with a one-hidden-layer feed-forward ReLU network with N hidden neurons. It has the form of a linear combination of ReLU basis functions like

$$f(x) = \sum_{i=1}^{N} a_i \mathrm{ReLU}(w_i x + b_i) + c, \quad \mathrm{ReLU}(z) = \max\{z, 0\},$$

where all parameters are scalars when approximating one-dimensional functions, *i.e.*, $w_i, b_i, a_i, c \in \mathbb{R}$. Since ReLU activation is positively homogeneous *i.e.*, $\mathrm{ReLU}(\lambda x) = \lambda \mathrm{ReLU}(x)$ for all $\lambda > 0$, we consider a homogeneous case with $w_i = \pm 1$. To facilitate our construction below, we assume N is an even number (*i.e.*, $N = 2n$) and the basis functions located pairwisely as

$$\phi_k^{\pm}(x) = \mathrm{ReLU}\big(\pm(x - b_k)\big), \quad k = 1, 2, \dots, n, \tag{1}$$

where the biases $\{b_k\}_{k=1}^n$ determine the basis locations.[1] The requirement of even N can be removed by adding a "ghost basis function" $\phi_k^{\pm}(x)$ with $a_i = 0$ if it can be paired with an existing $\phi_k^{\mp}(x)$ also with $a_i = 0$. Nevertheless, we retain this condition for simplicity. Next, we introduce two factors α, γ to adjust the network's output, leading to an additional one-dimensional linear layer. While this layer is not essential for achieving UAP, it does simplify the proof and offer practical value. The network's output function f^{NN} gives

$$f^{\mathrm{NN}}(x) = \alpha + \gamma \sum_{k=1}^{n} \big[p_k \phi_k^{+}(x) + q_k \phi_k^{-}(x)\big], \tag{2}$$

[1] In this paper, we use this notation $\{a_k\}_{k=1}^n := \{a_1, \dots, a_n\}$ to represent a set with n elements, while $(a_k)_{k=1}^n := (a_1, \dots, a_n)$ for a vector with n components.

where $\theta^{(2n)} = (p_1, q_1, \dots, p_n, q_n) \in [-1, 1]^{2n}$ are the coefficient vector of basis functions, which also correspond to the parameters in the second hidden layer of the network.

2.2. Permutation and corresponding properties

The permutation of a vector can be described as a process of rearranging the elements within, leaving the actual value unchanged. Concretely, it can be defined as:

Definition 2.1. For a vector $V^{(m)} = (v_1, v_2, \dots, v_m)$, the permutation τ is a bijection from the element set $\{v_i\}_{i=1}^m$ to itself.

We denote the permuted vector as $\tau(V^{(m)}) = (\tau(v_i))_{i=1}^m$. Notice that all permutations of $V^{(m)}$ can form a group S_m, which is closed under composition. Precisely, for any $\pi, \tau \in S_m$, there is a $\rho \in S_m$ such that $\rho(v_i) = \pi(\tau(v_i))$ for all v_i within $V^{(m)}$. This property leads to the major advance of permutation training: ideally, the final weight θ_T can be achieved by permuting the initialized weight θ_0 only once, *i.e.*, there exists a $\tau \in S_{2n}$ such that $\theta_T = \tau(\theta_0)$. This *one-step* nature significantly distinguishes permutation training from other iterative training methods like Adam.

2.3. Weight configuration and main theorems

We apply the permutation training on the second hidden layer's weights $\theta^{(2n)}$, leading to the following configuration: the coefficient vector $\theta^{(2n)}$ is permuted from a predetermined vector $W^{(2n)} \in \mathbb{R}^{2n}$, *i.e.*, $\theta^{(2n)} = \tau(W^{(2n)})$. Without loss of generality, we consider the target continuous function $f^* \in C([0, 1])$. Our results begin with a basic scenario with equidistantly distributed location vector $B^{(n)}$ and pairwise coefficient vector $W^{(2n)}$. The UAP of a permutation-trained network to f^* can be stated as follows:

Theorem 2.1 (*UAP With a Linear Layer*). *For any function $f^* \in C([0, 1])$ and any small number $\varepsilon > 0$, there exists a large integer $n \in \mathbb{Z}^+$, and $\alpha, \gamma \in \mathbb{R}$ for f^{NN} in Eq. (2) with equidistantly distributed $B^{(n)} = (b_i)_{i=1}^n := \left(0, \frac{1}{n-1}, \dots, 1\right)$ and corresponding $W^{(2n)} = (\pm b_i)_{i=1}^n := (+b_1, -b_1, \dots, +b_n, -b_n)$, along with a permuted coefficients $\theta^{(2n)} = \tau(W^{(2n)})$, such that $|f^{NN}(x) - f^*(x)| \le \varepsilon$ for all $x \in [0, 1]$.*

The intuition of this result comes from the rich expressive possibility of permutation training. Next, we enhance the result in Theorem 2.1 to a purely permuted situation, suggesting the UAP can be achieved without changing α, γ in Eq. (2).

Theorem 2.2 (*UAP Without the Linear Layer*). *Let $\alpha = 0, \gamma = 1$. For any function $f^* \in C([0, 1])$ and any small number $\varepsilon > 0$, there exists a large integer $n \in \mathbb{Z}^+$, for f^{NN} in Eq. (2) with equidistantly distributed $B^{(n)} = (b_i)_{i=1}^n := \left(0, \frac{1}{n-1}, \dots, 1\right)$ and $W^{(2n)} = (\pm b_i)_{i=1}^n := (+b_1, -b_1, \dots, +b_n, -b_n)$, along with a permuted coefficients $\theta^{(2n)} = \tau(W^{(2n)})$ such that $|f^{NN}(x) - f^*(x)| \le \varepsilon$ for all $x \in [0, 1]$.*

Although Theorem 2.2 considers a theoretically stronger setting, the additional requirement of n reveals the practical meanings of learnable α, γ in reducing the necessary network width to achieve UAP. Moreover, the result can be generalized to the scenario with pairwise random initialization, which is stated by the following theorem.

Theorem 2.3 (*UAP for Randomly Initialized Parameters*). *Given a probability threshold $\delta \in (0, 1)$, for any function $f^* \in C([0, 1])$ and any small number $\varepsilon > 0$, there exists a large integer $n \in \mathbb{Z}^+$, and $\alpha, \gamma \in \mathbb{R}$ for f^{NN} in Eq. (2) with randomly initialized $B_{rand}^{(n)} \sim \mathcal{U}[0, 1]^n$ and pairwisely randomly initialized $W_{rand}^{(2n)} = (\pm p_i)_{i=1}^n$, $p_i \sim \mathcal{U}[0, 1]$, along with a permuted coefficients $\theta^{(2n)} = \tau(W_{rand}^{(2n)})$, such that with probability $1 - \delta$, $|f^{NN}(x) - f^*(x)| \le \varepsilon$ for all $x \in [0, 1]$.*

2.4. Proof ideas

To identify the UAP of our network Eq. (2) in $C([0, 1])$, we employ a piecewise constant function, which is a widely-used continuous function approximator (Stein & Shakarchi, 2009), and can be expressed as a summation of several step functions. Next, we demonstrate that our networks can approximate each step function. In this spirit, our constructive proof includes the following three steps (illustrated in Fig. 1):

1. Approach the target function f^* by a piecewise constant function g;
2. Approximate each step functions of g by a subnetwork within f^{NN};
3. Annihilate the impact of the remaining parts of f^{NN}.

Step 1 can be achieved by cutting the range of f^* into subregions with an equal width Δh, and then constructing a step function at each point where f^* crosses a boundary of these subregions, leading to an approximator with an error Δh (illustrated in Fig. 1(a)). The statement is outlined below.

Lemma 2.1. *For any function $f^* \in C([0, 1])$ and any small number $\varepsilon' > 0$, there is a piecewise constant function g with a common jump $\Delta h \le \varepsilon'$, such that $|g(x) - f^*(x)| \le \varepsilon'$ for all $x \in [0, 1]$. Moreover, the function g can be written as a summation of J step functions $\{f_{s_j}\}_{j=1}^J$ as the following form,*

$$g(x) = \sum_{j=1}^{J} a_j f_{s_j}(x) = \sum_{j=1}^{J} a_j \Delta h \chi(x - s_j), \quad a_j = \pm 1,\ s_j \in [0, 1],\ J \in \mathbb{Z}^+, \tag{3}$$

Here J is the step number, s_j is the step location, a_j is the step sign controlling the direction, and χ is the standard step function

$$\chi(x) = \begin{cases} 0, & x < 0, \\ 1, & x \ge 0. \end{cases}$$

Proof. The function g can be constructed explicitly. According to the Stone–Weierstrass theorem (Stone, 1948), we assume f^* to be a polynomial function for simplicity. Let the range of f^* be covered by an interval $[k_{\min}\Delta h, k_{\max}\Delta h]$ with two integers $k_{\min}, k_{\max} \in \mathbb{Z}$. Then denote J as the number of intersections between f^* and parallel lines $y = (k + 0.5)\Delta h$, $k = k_{\min}, k_{\min} + 1, \dots, k_{\max} - 1$. Since f^* is a polynomial function, J can be assumed to be finite (Otherwise f^* will be a constant function due to the fundamental theorem of algebra (Hungerford, 2012), making it easily approximated by g). The scenario with $J = 0$ is also trivial since it indicates that f^* lies in $[(k_0 - 0.5)\Delta h, (k_0 + 0.5)\Delta h]$ for some integer $k_0 \in [k_{\min}, k_{\max}]$, such that f^* can be approached by the constant function $y = k_0 \Delta h$ with an error Δh.

Hence, for any $j = 1, \dots, J$, we choose $s_j \in [0, 1]$ such that $f^*(s_j) = (k_j + 0.5)\Delta h$ for some $k_j \in \mathbb{Z}$. The step locations $s_1 < \cdots < s_J$ are distinct since any repetition would contradict the continuity of the target function f^*. The step sign a_j is determined according to the values of f^* on $[s_{j-1}, s_{j+1}]$. It is easy to verify such construction satisfies our requirements. □

The execution of step 2 is inspired by the divide-and-conquer algorithm in computer science (Hopcroft, Ullman, & Aho, 1983). For each step function f_{s_j} in g, we select basis functions to construct a step function approximator $f_{s_j}^{\mathrm{NN}}$, then sum them up to approach g. This *step-matching* construction utilizes four pairs of basis functions $\{\pm b_i\}_{i=1}^4$ (shown in Fig. 1(b)), and establishing a one-to-one mapping between coefficients and biases, *i.e.*, $\{p_i, q_i\}_{i=1}^4 = \{\pm b_i\}_{i=1}^4$. It ensures that each coefficient and location are uniquely assigned and prevents conflict between different approximators.

Step 3 plays a vital role in the proof construction, and serves as an essential distinguishing factor that sets permutation training apart from conventional scenarios. Note that the specific setting of permutation

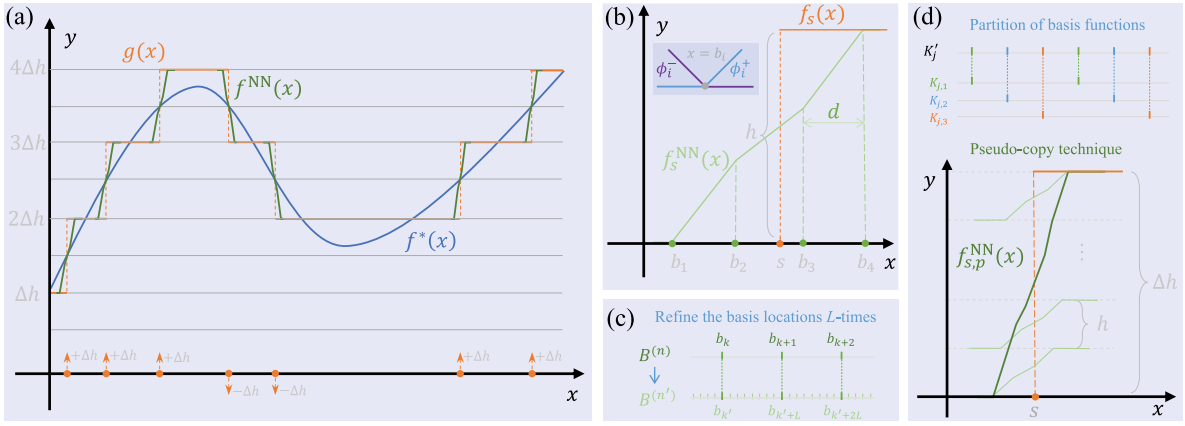


Fig. 1. Main idea of the construction. (a) Approximate the continuous function f^* by a piecewise constant function g which is further approximated by permuted networks f^{NN}. (b) The step function approximator f_s^{NN} constructed by step-matching. (c) Refine the basis functions L-times. (d) Stacking pseudo-copies to achieve the desired height.

training poses a crucial challenge that the proof must utilize every parameter, rather than just pick up the desired parameters and discard the rest. Therefore, it is essential to eliminate the remaining network parameters after step 2 to prevent the potential accumulation of errors. We solve this problem by proposing a *linear reorganization* to write the remaining part as a linear function with a controllable slope.

To further enhance the conclusion of Theorem 2.1 to Theorem 2.2, we first introduce a technique called *pseudo-copy*, which can achieve UAP without altering γ in Eq. (2). By partitioning the refined basis functions, several pseudo-copies $f_{s,p}^{\mathrm{NN}}$ of the original approximator f_s^{NN} can be constructed with a controllable error. The final height can then be achieved by stacking these copies together rather than changing γ (see Fig. 1(d)). Additionally, to make the shift factor α removable, our *constant-matching* construction can provide the necessary shifting. It also enables another way to eliminate the remaining part of the network.

Extending the UAP to the random initializations is justified by the fact that the parameters randomly sampled from uniform distributions become denser, thus approaching the equidistant case. Therefore, a sufficiently wide network has a high probability of finding a subnetwork that is close enough to the network with UAP in the equidistant case. Then this subnetwork can also achieve UAP due to its continuity. The remaining part of the network can be eliminated by step 3.

3. UAP of permutation-trained networks

This section provides a detailed construction of the approximator with weight-permuted networks in the equidistant case, along with an estimation of the convergent rate of approximation error. The extension to the scenario with random initialization is also thoroughly discussed.

3.1. The construction of step, constant, and linear function approximators

Lemma 2.1 enables us to approximate the target function f^* by a piecewise constant function g in Eq. (3). Next, we aim to approximate each step function f_{s_j} within g, respectively, and then eliminate the remaining part of the network. This section introduces several key constructions to achieve this goal in the equidistant case.

3.1.1. Step-matching construction of step function approximators f_s^{NN}

Here we construct the step function approximator f_s^{NN} for a given step function $f_s(x) = \Delta h\,\chi(x-s)$ with height Δh and location s. The construction considers four pairs of basis functions $\{\phi_i^{\pm}\}_{i=1}^4$ with locations $\{b_i\}_{i=1}^4$ and coefficients $\{p_i,q_i\}_{i=1}^4 = \{\pm b_i\}_{i=1}^4$, which has the following form,

$$f_s^{\mathrm{NN}}(x) = \sum_{i=1}^4 p_i\phi_i^+(x) + \sum_{i=1}^4 q_i\phi_i^-(x), \quad x\in[0,1]. \tag{4}$$

Here we require the locations $b_1 < b_2 < b_3 < b_4$ satisfy the following symmetric condition,

$$d := b_2 - b_1 = b_4 - b_3 \implies b_1 + b_4 = b_2 + b_3, \tag{5}$$

where d is the basis distance. To ensure a local error of the approximator, we appeal f_s^{NN} to be x-independent outside the interval $[b_1,b_4]$. As a result, the coefficients p_i,q_i must satisfy $\sum_{i=1}^4 p_i = \sum_{i=1}^4 q_i = 0$, which implies the correspondence between $\{p_i,q_i\}_{i=1}^4$ and $\{\pm b_i\}_{i=1}^4$ as

$$\begin{aligned} &p_1 = -b_1, \quad p_2 = +b_2, \quad p_3 = +b_3, \quad p_4 = -b_4,\\ &q_1 = +b_4, \quad q_2 = -b_3, \quad q_3 = -b_2, \quad q_4 = +b_1, \end{aligned} \tag{6}$$

We call the $\{p_i,q_i\}_{i=1}^4$ and $\{\pm b_i\}_{i=1}^4$ is *step-matching* if they satisfy Eq. (6), which gives the piecewise form of f_s^{NN} in Eq. (4) as

$$f_s^{\mathrm{NN}}(x) = \begin{cases} 2b_1b_4 - 2b_2b_3 & 0\le x < b_1,\\ (-b_1+b_4)x + b_1^2 + b_1b_4 - 2b_2b_3 & b_1\le x < b_2,\\ (-2b_1+2b_2)x + b_1^2 - b_2^2 + b_1b_4 - b_2b_3 & b_2\le x < b_3,\\ (-b_1+b_4)x + b_1^2 - b_2^2 - b_3^2 + b_1b_4 & b_3\le x < b_4,\\ b_1^2 - b_2^2 - b_3^2 + b_4^2 & b_4\le x\le 1. \end{cases} \tag{7}$$

The profile of this f_s^{NN} can be found in Fig. 1(b), which shows that f_s^{NN} is monotone and can approach a step function with the height h satisfying the following relation,

$$h = 2(b_1^2 - b_2^2 - b_3^2 + b_4^2) = 4d(b_4 - b_2). \tag{8}$$

Notice that choosing the basis functions adjacently will lead to $d = \frac{1}{n-1}$ and $h = 8d^2$. By shifting $h/2$ and scaling $\Delta h/h$, we use f_s^{NN} to approach step function f_s with $s\in[b_1,b_4]$. It is obvious that the L^∞ error has the following trivial bound,

$$\left|\frac{\Delta h}{h}\left[f_s^{\mathrm{NN}}(x) + \frac{h}{2}\right] - f_s(x)\right| \le \Delta h, \quad \forall\, x\in[0,1]. \tag{9}$$

Eq. (7) implies that the approximation is exactly accurate when $x\notin[b_1,b_4]$. A toy example of this step-matching construction is shown in Appendix F.

Remark 1. Although Cai, Chen, and Liu (2021) suggested that a step function could be approached by two ReLU basis functions, this approach is unsuitable for the permutation training scenario. For an error tolerance ε, a step function $f_s(x) = h_s\chi(x-s)$ can be well approximated by a linear combination of two ReLU basis functions as

$$\tilde{f}_s^{\mathrm{NN}}(x) = \frac{h_s}{2\varepsilon}[\mathrm{ReLU}(x-s+\varepsilon) - \mathrm{ReLU}(x-s-\varepsilon)].$$

However, the dependence of the coefficients on the step height h_s hinders further construction since each coefficient can be used only once.

3.1.2. Constant-matching construction of constant function approximators f_c^{NN}

The four-pair form in Eq. (4) can also be utilized to approximate a constant function, which plays a crucial role in the proof of Theorem 2.2. The constant function approximator (denoted as f_c^{NN}) shares the same form of f_s^{NN} in Eq. (4) but differs slightly in the parameter assignment in Eq. (6). The coefficients $\{p_i, q_i\}_{i=1}^4$ are set to equalize the height of the two constant pieces $x < b_1$ and $b_4 \le x$, leading to $-\sum_{i=1}^4 p_i b_i = \sum_{i=1}^4 q_i b_i$. A possible choice is

$$\begin{aligned} &p_1 = -b_1, \quad p_2 = +b_2, \quad p_3 = +b_3, \quad p_4 = -b_4, \\ &q_1 = +b_1, \quad q_2 = -b_2, \quad q_3 = -b_3, \quad q_4 = +b_4. \end{aligned} \tag{10}$$

We call the $\{p_i, q_i\}_{i=1}^4$ and $\{\pm b_i\}_{i=1}^4$ is *constant-matching* if they satisfy Eq. (10). With these chosen coefficients $\{p_i, q_i\}_{i=1}^4$ and the symmetry of coefficients in Eq. (5), the constant function approximator f_c^{NN} can be written as:

$$\begin{aligned} f_c^{\mathrm{NN}}(x) &= (b_2 + b_3 - b_1 - b_4)\, x + b_1^2 - b_2^2 - b_3^2 + b_4^2 \\ &= 2d(b_4 - b_2) = h/2, \quad x \in [0,1]. \end{aligned} \tag{11}$$

By the relations of h in Eq. (8), it gives a representation of constant $C = h/2$ without the approximation error, i.e., $|f_c^{\mathrm{NN}}(x) - h/2| = 0$ for all $x \in [0,1]$.

Furthermore, by changing the sign of $\{p_i, q_i\}_{i=1}^4$ in Eq. (10) simultaneously, an approximator f_{-c}^{NN} with a negative constant $-C = -h/2$ can also be constructed:

$$f_{-c}^{\mathrm{NN}}(x) = -2d(b_4 - b_2) = -h/2, \quad x \in [0,1].$$

Therefore, we can freely adjust the sign $a_j = \pm 1$ of the approached constant. It also enables us to construct f_c^{NN} and f_{-c}^{NN} with the same constant separately, then pair them up to offset with each other, *i.e.*, $f_c^{\mathrm{NN}}(x) + f_{-c}^{\mathrm{NN}}(x) = 0$ for all $x \in [0,1]$. A toy example of this constant-matching construction is shown in Appendix F.

3.1.3. Linear reorganization of the linear function approximators f_ℓ^{NN}

Apart from the four-pair construction discussed before, we can also construct a linear function f_ℓ^{NN} by reorganizing each pair of basis functions. Concretely, we choose a pair of basis functions located at b_i and set the coefficients as $\{p_i, q_i\} = \{\pm b_i\}$. The linear function approximator f_ℓ^{NN} can be written as the following form,

$$f_\ell^{\mathrm{NN}}(x) = m_i \ell_i, \quad \ell_i(x) := b_i \phi_i^+(x) - b_i \phi_i^-(x) = b_i x - b_i^2, \quad x \in [0,1], \tag{12}$$

where $m_i = \pm 1$ is a freely adjusted sign. We call this method as *linear reorganization*. It allows us to adjust the number of the total basis functions, as the L^∞-norm of f_ℓ^{NN} located at b_i has an upper bound of b_i^2. Hence we can choose the unwanted b_i that is small enough, then omit it along with the related coefficients $\{p_i, q_i\} = \{\pm b_i\}$ from the network without affecting the approximation error.

3.2. Annihilate the unused part of the network

After constructing approximators to approach the target function, the permutation training setting requires that the remaining parameters be suitably arranged to eliminate their impact. Notice that a pair of basis functions $\phi_i^\pm$ are either used together or not at all. We apply the linear reorganization in Eq. (12) to form them into linear functions $f_{\ell_i}^{\mathrm{NN}}$. Concretely, denote $I_{\mathrm{un}} \subset \{1, 2, \dots, n\}$ as the index of unused basis functions and $\bar{n}$ as the number of elements in I_{un}, which can be processed to be an even number. After taking a sum of all $i \in I_{\mathrm{un}}$, the resulting S_ℓ has the following linear form,

$$S_\ell(x) = \sum_{i \in I_{\mathrm{un}}} f_{\ell_i}^{\mathrm{NN}}(x) = \sum_{i \in I_{\mathrm{un}}} m_i b_i x - \sum_{i \in I_{\mathrm{un}}} m_i b_i^2 =: \beta x + \eta, \quad x \in [0,1], \tag{13}$$

where β is the slope and η is the intercept. The goal then is to control the L^∞-norm of S_ℓ. We first choose a proper $m_i \in \{-1, 1\}$ for each $i \in I_{\mathrm{un}}$ to reduce $|\beta|$, which is equivalent to assigning addition and subtraction operations within a given sequence to reduce the final result's absolute value. Motivated by the *Leibniz's test* (known as *alternating series test*) (Rudin, 1976), the following lemma offers a solution with an upper bound of β.

Lemma 3.1. *For an even number $\bar{n}$ and a sequence of real number $c_i \in \mathbb{R}, i = 1, \dots, \bar{n}$, there exists a choice of $m_i \in \{-1, 1\}$, $i = 1, \dots, \bar{n}$, such that*

$$0 \le \sum_{i=1}^{\bar{n}} m_i c_i \le \Delta c, \quad \Delta c = \max_{\substack{1 \le j \le \bar{n} \\ j \notin \operatorname{argmax} c_j}} \left(\min_{\substack{i \ne j \\ c_i \ge c_j}} (c_i - c_j) \right), \tag{14}$$

where Δc can be regarded as the largest gap between the adjacent elements after sorting c_i in descending order (see Appendix G for a toy example).

Proof. Without loss of generality, we assume $c_1 \ge c_2 \ge \cdots \ge c_{\bar{n}} \ge 0$ and thus $\Delta c = \max_i |c_i - c_{i-1}|$. Since $\bar{n}$ is an even number, we define a new series $\{r_j\}_{j=1}^{\bar{n}/2}$ as the difference between each pair of elements in $\{c_i\}_{i=1}^{\bar{n}}$,

$$r_j = c_{2j-1} - c_{2j} \ge 0, \quad j = 1, 2, \dots, \bar{n}/2.$$

Then we permute $(r_j)_{j=1}^{\bar{n}/2}$ to be descending order. Concretely, we find a permutation τ and denote $r'_j = \tau(r_j) = [\tau(c_{2j-1}) - \tau(c_{2j})]$, such that $r'_1 \ge r'_2 \ge \cdots \ge r'_{\bar{n}/2} \ge 0$. Next, we alternatively use addition and subtraction operations on each r'_j by noting $\lambda_j = (-1)^{j-1}$, $j = 1, \dots, \bar{n}/2$, and estimate the summation $S_{\bar{n}/2}$ as the following forms,

$$S_{\bar{n}/2} := \sum_{j=1}^{\bar{n}/2} \lambda_j r'_j = \begin{cases} (r'_1 - r'_2) + (r'_3 - r'_4) + \cdots \ge 0, \\ r'_1 - (r'_2 - r'_3) - (r'_4 - r'_5) - \cdots \le r'_1 \le \Delta c. \end{cases}$$

Note that $S_{\bar{n}/2} = \sum_{j=1}^{\bar{n}/2} \lambda_j [\tau(c_{2j-1}) - \tau(c_{2j})]$ is of the form of $\sum_{i=1}^{\bar{n}} m_i c_i$, hence the choice of m_i implied by λ_j satisfies our requirement and the proof is finished. □

Remark 2. Leibniz's test guarantees that an alternating series that decreases in absolute value is bounded by the leading term. However, we emphasize that utilizing Leibniz's test directly to $\{c_i\}_{i=1}^{\bar{n}}$ can only get a trivial bound. Therefore, the introduction of r_j is important to get the desired result.

After applying Lemma 3.1 to control the slope β, the intercept η is consequently determined by the chosen $\{m_i\}_{i \in I_{\mathrm{un}}}$. Thus, we can choose a constant to adjust the intercept, which establishes an upper bound for the error S_ℓ of the remaining part.

3.3. Proof of Theorem 2.1

Lemma 2.1 offers a piecewise constant function g to approach the target function f^*. Now we prove that by permuting the selected coefficients and eliminating the remaining part, f^{NN} can approximate piecewise constant function g with the same accuracy, enabling us to prove Theorem 2.1.

Proof of Theorem 2.1. For any target function $f^* \in C([0,1])$ and small number ε, we aim to construct a network f^{NN} with n pairs of basis functions to approximate f^* with error ε. The goal is achieved with the following points:

(a) Approach f^* by a piecewise constant function g. Employing Lemma 2.1 by letting $\varepsilon' = \varepsilon/4$, we can construct a piecewise constant function g in Eq. (3) with a constant height $\Delta h \le \varepsilon' = \varepsilon/4$, distinct step locations $s_1 < \cdots < s_J$. It gives $|g(x) - f^*(x)| \le \Delta h < \varepsilon/2$ for all $x \in [0,1]$. Here we denote $\delta_s := \max_j |s_j - s_{j-1}|$ as the maximal gap of step locations, where $j = 0, 1, \dots, J+1$ and $s_0 = 0$, $s_{J+1} = 1$.

(b) Approximate each step function in g by the step-matching approximator $f_{s_j}^{\mathrm{NN}}$ in Eq. (4). Since the step locations $\{s_j\}_{j=1}^J$ are distinct, we can choose a network f^{NN} with large enough $\hat{n}$, *i.e.*, the locations in $B^{(\hat{n})}$ are dense enough, such that there are enough basis functions to

construct $f_{s_j}^{\mathrm{NN}}$ for each $a_j f_{s_j}$, respectively. In fact, for any $\hat{n} > 8/\delta_s + 1$, the distance between basis functions $\hat{d} = 1/(\hat{n}-1)$ satisfies $\hat{d} < \delta_s/8$.

However, to maintain the consistency of the following estimation, we refine the basis locations $B^{(\hat{n})}$ to $B^{(n)}$ with $n = L(\hat{n}-1)+1$ for some integer $L \geq \Delta h(\hat{n}-1)^2/8$ (see Fig. 1(c)). Denote $K = \{1, \dots, n\}$ as the index set of the locations $B^{(n)}$, and for each $j = 1, \dots, J$, we choose the basis functions with the index $K_j := \{k_j, k_j + L, k_j + 2L, k_j + 3L\} \subset K$, such that $s_j \in [b_{k_j+L}, b_{k_j+2L})$ and $\{K_j\}_{j=1}^J$ has empty intersection. Therefore, for each $a_j f_{s_j}$, an approximator $(f_{s_j}^{\mathrm{NN}} + a_j h/2)$ with a shifting $a_j h/2$ can be constructed by applying the step-matching construction on $\{p_k, q_k\}_{k\in K_j}$. This approximation has the following error estimation given by Eq. (9):

$$\left| \left[f_{s_j}^{\mathrm{NN}}(x) + a_j \frac{h}{2} \right] - a_j \frac{h}{\Delta h} f_{s_j}(x) \right| \leq h, \quad \forall\, x \in [0,1], \quad j = 1, \dots, J, \tag{15}$$

where $h = 8\hat{d}^2 = 8/(\hat{n}-1)^2 = 8L^2/(n-1)^2$ is the height determined by $\hat{n}$, and the scaling $h/\Delta h$ serves to match the constant height Δh and its approximator height h. Our requirement of L gives the condition $L \geq \Delta h/h$. Furthermore, denote $K_{\mathrm{use}} = \cup_{j=1}^J K_j$ as the index set for all involved basis functions in Eq. (15), which leads to the requirement of $\hat{n} > 4J$. We define S_{use} as the picked subnetwork of f^{NN},

$$S_{\mathrm{use}}(x) := \sum_{k\in K_{\mathrm{use}}} \left[p_k \phi_k^+(x) + q_k \phi_k^-(x) \right] = \sum_{j=1}^J f_{s_j}^{\mathrm{NN}}(x), \quad x \in [0,1],$$

then the properties of our step-matching construction and Eq. (15) imply the error E_{use} of the summed approximators can be estimated as the following form,

$$\begin{aligned} E_{\mathrm{use}} &:= \max_{x\in[0,1]} \left| S_{\mathrm{use}}(x) + \frac{h}{2} \sum_{j=1}^J a_j - \frac{h}{\Delta h} g(x) \right| \\ &= \max_{x\in[0,1]} \left| \sum_{j=1}^J \left[f_{s_j,p_l}^{\mathrm{NN}}(x) + a_j \frac{h}{2} \right] - \sum_{j=1}^J \frac{h}{\Delta h} a_j f_{s_j}(x) \right| \leq h. \end{aligned} \tag{16}$$

Here $\sum_{j=1}^J a_j =: J' \leq J$ is a constant, and height h satisfies $h = 8L^2/(n-1)^2$.

(c) Annihilate the impact of the unused part. The linear reorganization in Eq. (12) enables write the unused part in f^{NN} into a linear function S_{un}, namely,

$$S_{\mathrm{un}}(x) = \sum_{k\in K\setminus K_{\mathrm{use}}} \left[p_k \phi_k^+(x) + q_k \phi_k^-(x) \right] = \sum_{k\in K\setminus K_{\mathrm{use}}} (m_k b_k x - m_k b_k^2) =: \beta x + \eta,$$

Since the number of unused basis functions can be processed to be even, we apply Lemma 3.1 on the series $\{b_k\}_{k\in K\setminus K_{\mathrm{use}}}$ to get a choice of $m_k \in \{-1, 1\}$ for each $k \in K \setminus K_{\mathrm{use}}$, which provides an upper bound of the slope β as $0 \leq \beta \leq \Delta b$, where Δb is the largest gap between the adjacent locations in $\{b_k\}_{k\in K\setminus K_{\mathrm{use}}}$. Notice that in f^{NN}, the refined basis distance $d < 1/L\hat{n}$ is small enough to ensure that there is at least one unused basis function between two adjacent used basis functions, *i.e.*, $\Delta b \leq 2/L\hat{n}$. To control the error of the intercept η, we introduce a shifting $C_\eta = -\eta$, thus the error E_{un} introduced by the unused part gives the following estimation,

$$E_{\mathrm{un}} := \max_{x\in[0,1]} \left| S_{\mathrm{un}}(x) + C_\eta \right| \leq \Delta b < \frac{2}{L\hat{n}} \leq \frac{2h}{\Delta h \hat{n}}. \tag{17}$$

Therefore, we choose $\hat{n} > 2/\Delta h$ such that the error E_{un} satisfies $E_{\mathrm{un}} \leq h$.

(d) Complete the proof by choosing the suitable values of γ, α. We choose $\hat{n}$ and two factors γ, α, such that

$$\hat{n} \geq \max\left\{ 4J, \frac{8}{\delta_s} + 1, \frac{2}{\Delta h} \right\}, \quad \gamma = \frac{\Delta h}{h}, \quad \alpha = \Delta h \left(\frac{J'}{2} + \frac{C_\eta}{h} \right).$$

Moreover, Let $n = L(\hat{n}-1)+1$ for some integer $L \geq \Delta h(\hat{n}-1)^2/8$, then Eq. (17)–(19) implies that the approximation error of the network f^{NN} with width n satisfies

$$\begin{aligned} \left| f^{\mathrm{NN}}(x) - g(x) \right| &= \left| \gamma \sum_{k\in K} \left[p_k \phi_k^+(x) + q_k \phi_k^-(x) \right] + \alpha - g(x) \right| \\ &\leq \left| \frac{\Delta h}{h} \left[S_{\mathrm{use}}(x) + S_{\mathrm{un}}(x) \right] + \left(\frac{\Delta h}{2} J' + \frac{\Delta h}{h} C_\eta \right) - g(x) \right| \\ &\leq \frac{\Delta h}{h} \left| S_{\mathrm{use}}(x) + \frac{h}{2} J' - \frac{h}{\Delta h} g(x) \right| + \frac{\Delta h}{h} \left| S_{\mathrm{un}}(x) + C_\eta \right| \\ &\leq \frac{\Delta h}{h} (E_{\mathrm{use}} + E_{\mathrm{un}}) \leq 2\Delta h \leq \varepsilon/2, \quad \forall\, x \in [0,1]. \end{aligned}$$

This complete the proof since $|f^{\mathrm{NN}}(x) - f^*(x)| \leq \varepsilon$ for all $x \in [0,1]$. □

3.4. Proof of Theorem 2.2

Next, we remove scaling γ and shifting α during the proof, *i.e.,* achieve UAP with fixed factors $\gamma = 1, \alpha = 0$. To eliminate the scaling γ, we introduce the *pseudo-copy* technique to let $\gamma = L$ for an integer $L \in \mathbb{Z}^+$, allowing copy the approximator L-times and stack them (instead of multiplying by γ) to match the desired height. The shifting α can be replaced by our constant-matching construction in Eq. (11), which is also applied to eliminate the remaining parameters, since our linear reorganization in Eq. (12) requires a shifting C_η to control the error.

Proof of Theorem 2.2. For a given target function $f^* \in C([0,1])$, we first apply Theorem 2.1 to obtain a network f^{NN} to approximate f^*, and then enhance the construction by pseudo-copy technique and constant-matching construction to eliminate the scaling γ and shifting α, leading to a network f^{NN} in Eq. (2) with fixed factors $\gamma = 1, \alpha = 0$. To facilitate distinction, we will denote this network equipped with pseudo-copy technique as $f_{\mathrm{pc}}^{\mathrm{NN}}$ in the following text.

(a) Approximate the target function f^* by a network f^{NN} with learnable factors γ and α. By applying Theorem 2.1, we construct f^{NN} to approximate f^* through a piecewise constant function g with error ε. Following the previous discussion, we have $\Delta h \leq \varepsilon/8$, $\gamma = \Delta h/h$. During the construction, L are chosen to be $L = \gamma$.

(b) Remove the scaling γ by the pseudo-copy technique. We first reassign the basis functions in f^{NN} to construct the pseudo-copies of each approximator $f_{s_j}^{\mathrm{NN}}$. For each $K_j = \{k_j, k_j+L, k_j+2L, k_j+3L\} \subset K$, we denote the corresponding adjacent index set for pseudo-copy as $K_j' := \{k_j, k_j+1, \dots, k_j+4L-1\}$ such that $K_j \subset K_j'$. To maintain the same height of each pseudo-copy, we partition K_j' into L subsets as $K_j' = \cup_{l=1}^L K_{j,l}$ by choosing after every L indexes (illustrated at the top of Fig. 1(d)).

For each $l = 1, \dots, L$, we apply the step-matching construction on $\{p_k, q_k\}_{k\in K_{j,l}}$ to construct the pseudo-copy $f_{s_j,p_l}^{\mathrm{NN}}$ for $f_{s_j}^{\mathrm{NN}}$ in Eq. (15), respectively. Since each $f_{s_j,p_l}^{\mathrm{NN}}$ has the same height h with $f_{s_j}^{\mathrm{NN}}$, we have the copy error as $|f_{s_j,p_l}^{\mathrm{NN}}(x) - f_{s_j}^{\mathrm{NN}}(x)| < h$ for all $x \in [0,1]$. Therefore, Eq. (15) allows the summed $f_{s_j,p_l}^{\mathrm{NN}}$ to approximate $a_j f_{s_j}$ with the following error:

$$\begin{aligned} &\left| \sum_{l=1}^L \left[f_{s_j,p_l}^{\mathrm{NN}}(x) + a_j \frac{h}{2} \right] - a_j f_{s_j}(x) \right| \\ &\leq \sum_{l=1}^L \left| f_{s_j,p_l}^{\mathrm{NN}}(x) - f_{s_j}^{\mathrm{NN}}(x) \right| + \left| L \left[f_{s_j}^{\mathrm{NN}}(x) + a_j \frac{h}{2} \right] - a_j f_{s_j}(x) \right| \\ &\leq Lh + \Delta h = 2\Delta h, \quad \forall\, x \in [0,1], \quad j = 1, \dots, J. \end{aligned} \tag{18}$$

Here $a_j h/2$ is the shifting required by the pseudo-copies.

(c) Replace the shifting with constant-matching construction. After constructing the pseudo-copies $f_{s_j,p_l}^{\mathrm{NN}}$, our constant-matching construction in Eq. (11) can pair them up with the necessary shifting $f_{c_j}^{\mathrm{NN}} = a_j h/2$, enabling the combined $\sum_{l=1}^L (f_{s_j,p_l}^{\mathrm{NN}} + f_{c_j}^{\mathrm{NN}})$ to approach $a_j f_{s_j}$. Denote K_{use}' as the index set for all involved basis functions. We define

S'_{use} as the picked subnetwork of $f_{\text{pc}}^{\text{NN}}$,

$$S'_{\text{use}}(x) := \sum_{k\in K'_{\text{use}}} \left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] = \sum_{l=1}^{L}\sum_{j=1}^{J}\left[f_{s_j,p_l}^{\text{NN}}(x) + f_{c_j}^{\text{NN}}(x)\right].$$

where $x \in [0,1]$. Eq. (18) implies the following error estimation,

$$\begin{aligned} E'_{\text{use}} &:= \max_{x\in[0,1]} \left|S'_{\text{use}}(x) - g(x)\right| \\ &= \max_{x\in[0,1]} \left|\sum_{j=1}^{J}\sum_{l=1}^{L}\left[f_{s_j,p_l}^{\text{NN}}(x) + f_{c_j}^{\text{NN}}(x)\right] - \sum_{j=1}^{J} a_j f_{s_j}(x)\right| \le 2\Delta h. \end{aligned} \tag{19}$$

This construction uses $8L$ basis locations, leading to the requirement as $n > 8LJ$.

(d) Eliminate the unused part of the network. Since the linear reorganization in Eq. (12) is invalid, we turn to apply the constant-matching construction in Eq. (11) to eliminate the unused part of the network. Concretely, for the remaining $n - 8JL$ pairs of basis functions in $f_{\text{pc}}^{\text{NN}}$, we construct $f_{\pm c_t}^{\text{NN}}$, $t = 1, \ldots, \lfloor n/8\rfloor - JL$, then pairing them up to offset with each other, *i.e.*,

$$S'_{\text{un}}(x) = \sum_{k\in K\setminus K'_{\text{use}}} \left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] = \sum_{t=1}^{\lfloor n/8\rfloor - JL}\left[f_{c_t}^{\text{NN}}(x) + f_{-c_t}^{\text{NN}}(x)\right] + R(x),$$

Here $R(x)$ is the residual part since n is not always divisible by 8. However, our linear reorganization in Eq. (12) enables omit the basis functions with sufficiently small b_i. Concretely, the L^∞ error introduced by $R(x)$ can be controlled as $|R(x)| \le \sum_{k=1}^{4}\bar{b}_k^2 =: C_R$ for all $x \in [0,1]$, where $\{\bar{b}_k\}_{k=1}^4$ is the missed or external basis locations. Since the constant-matching construction has flexibility, $\{\bar{b}_k\}_{k=1}^4$ can be small enough to ensure $C_R \le \Delta h$. Hence, the error E'_{un} introduced by the unused parameters satisfies

$$E'_{\text{un}} = \max_{x\in[0,1]} \left|S'_{\text{un}}(x)\right| = C_R + C_c \le 2\Delta h. \tag{20}$$

Here the constant C_c comes from the possible mismatch in pairing up $f_{c_t}^{\text{NN}} + f_{-c_t}^{\text{NN}}$. However, for a sufficiently wide network, C_c can be small enough as $C_c < \Delta h$.

(e) Complete the proof by combining the previous estimation. By setting $\gamma = 1, \alpha = 0$ and choosing a sufficiently large $L \in \mathbb{Z}^+$, such that

$$n = \sqrt{\frac{8L^3}{\Delta h}} + 1 > \max\left\{8JL,\ \frac{8L}{\delta_s} + L,\ \frac{2L}{\Delta h}\right\}.$$

Eq. (19)–(20) gives estimation of the overall approximation error as follows:

$$\begin{aligned} \left|f_{\text{pc}}^{\text{NN}}(x) - g(x)\right| &= \left|\sum_{k\in K}\left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] - g(x)\right| \\ &\le \left|S'_{\text{use}}(x) - g(x)\right| + \left|S'_{\text{un}}(x)\right| \\ &\le E'_{\text{use}} + E'_{\text{un}} \le 4\Delta h, \quad \forall x \in [0,1]. \end{aligned}$$

where Δh is chosen to satisfy $\Delta h \le \varepsilon/8$. Hence we can finish the proof of Theorem 2.2 since $|f_{\text{pc}}^{\text{NN}}(x) - f^*(x)| \le \varepsilon$ for all $x \in [0,1]$. □

3.5. Estimate the approximation rate

Here we estimate the approximation rate of the L^2-error e_s of approximating single step function f_s in Eq. (3) by the approximator f_s^{NN} in Eq. (7). In our four-pair construction, we assume $s = (b_2 + b_3)/2$ and introduce k_1 and k_2 to rewrite the symmetry relations in Eq. (5) as:

$$\begin{aligned} b_1 &= s - k_2, \quad b_3 = s + k_1, \\ b_2 &= s - k_1, \quad b_4 = s + k_2, \end{aligned} \tag{21}$$

where $0 < k_1 \le k_2$, and it also gives $d = k_2 - k_1$. The piecewise form of step function approximator f_s^{NN} in Eq. (7) enables us to subdivide the error into parts like

$$\begin{aligned} e_s^2 &= \int_0^1 \left|\gamma\left[f_s^{\text{NN}}(x) + \frac{h}{2}\right] - \frac{h}{\Delta h} f_s(x)\right|^2 \mathrm{d}x \\ &= \gamma^2\left[\int_{b_1}^{s}\left|f_s^{\text{NN}}(x) + \frac{h}{2}\right|^2 \mathrm{d}x + \int_{s}^{b_4}\left|f_s^{\text{NN}}(x) - \frac{h}{2}\right|^2 \mathrm{d}x\right]. \end{aligned} \tag{22}$$

After calculation, the error of each approximator f_s^{NN} can be estimated like

$$e_s^2 = \gamma^2\left[\frac{8}{3}(k_1 - k_2)^2\left(k_1^3 + 3k_1^2k_2 + 2k_1k_2^2 + k_2^3\right)\right] \le \gamma^2\frac{56}{3}d^2k_2^3. \tag{23}$$

During the construction of f_s^{NN}, the basis functions are chosen adjacently, leading to $k_2 \sim \mathcal{O}(d)$ and $e_s \sim \mathcal{O}(\gamma\, d^{\frac{5}{2}})$. To estimate the order of γ, we derive from Eq. (8) that the height h gives $h \sim \mathcal{O}(d^2)$, while the step function f_s has a d-independent height $\Delta h \sim \mathcal{O}(1)$. Therefore, the scaling $\gamma = \Delta h/h \sim \mathcal{O}(d^{-2})$ is needed, and the error is rewritten as $e_s \sim \mathcal{O}(d^{\frac{1}{2}})$. Recall that d in Eq. (5) has $d \sim \mathcal{O}(\frac{1}{n-1})$, we have $e_s \sim \mathcal{O}(n^{-\frac{1}{2}})$, which means the approximation rate is roughly $1/2$ order with respect to the network width n. We will numerically verify this rate in Section 4.

This estimation also holds for our pseudo-copy f_{s_j,p_l}^{NN}, where $\gamma = L$. The triangle inequality is adopted to estimate the overall approximation error of these stacked $\{f_{s,p_l}^{\text{NN}}\}_{l=1}^{L}$ to a predetermined step function f_s, *i.e.*,

$$e_{s,p} = \left\|\sum_{l=1}^{L}\left(f_{s,p_l}^{\text{NN}} + \frac{h}{2}\right) - f_s\right\|_{L^2} \le \sum_{l=1}^{L}\left\|\left(f_{s,p_l}^{\text{NN}} + \frac{h}{2M}\right) - \frac{1}{L}f_s\right\|_{L^2} =: \sum_{l=1}^{L} e_{s,p_l}. \tag{24}$$

Now we focus on the approximation error e_{s,p_l} of each f_{s,p_l}^{NN} to the f_s/L. However, the result in Eq. (23) cannot be directly adopted since it only holds for the symmetry case in Eq. (21). Instead, we choose locations $\{\tilde{b}_i\}_{i=1}^4$ *almost* symmetrically with mismatch measured by Δs_l. Therefore, the relation in Eq. (21) is transformed into

$$\begin{aligned} \tilde{b}_1 &= (s + \Delta s_l) - k_2, \quad \tilde{b}_3 = (s + \Delta s_l) + k_1, \\ \tilde{b}_2 &= (s + \Delta s_l) - k_1, \quad \tilde{b}_4 = (s + \Delta s_l) + k_2. \end{aligned} \tag{25}$$

Compared with Eq. (21), it is clear that this transformation is equivalent to replacing $f_{s,p_l}^{\text{NN}}(x)$ with $f_s^{\text{NN}}(x - \Delta s_l)$ for all $x \in [0,1]$. Therefore, each e_{s,p_l} in Eq. (24) gives

$$\begin{aligned} e_{s,p_l}^2 &= \int_0^1 \left|\left[f_{s_i}^{\text{NN}}(x - \Delta s_l) + \frac{h}{2}\right] - \frac{1}{L}f_s(x)\right|^2 \mathrm{d}x \\ &= \int_{b_1+\Delta s_l}^{s}\left|f_{s_i}^{\text{NN}}(x - \Delta s_l) + \frac{h}{2}\right|^2 \mathrm{d}x + \int_{s}^{b_4+\Delta s_l}\left|f_{s_i}^{\text{NN}}(x - \Delta s_l) - \frac{h}{2}\right|^2 \mathrm{d}x, \end{aligned} \tag{26}$$

where the integral range $[b_1 + \Delta s_l, b_4 + \Delta s_l]$ is not symmetrical about $x = s$ due to the mismatch. However, since Δs_l is small, we assume that $b_2 + \Delta s_l \le s \le b_3 + \Delta s_l$, then follow the similar procedure used in Eq. (22) to divide the error in Eq. (26) into parts. After calculation, we obtain the following estimation

$$\begin{aligned} e_{s,p_l}^2 &= \frac{8}{3}(k_1 - k_2)^2\left[k_1^3 + 3k_1^2k_2 + 2k_1k_2^2 + k_2^3 + 3\Delta s_l^2(k_1 + k_2)\right] \\ &= \frac{8}{3}d^2\left(-d^3 + 6d^2k_2 - 11dk_2^2 + 7k_2^3 - 3d\,\Delta s_l^2 + 6k_2\,\Delta s_l^2\right). \end{aligned}$$

Since Δs_l can be assumed to be small as $\Delta s_l \sim \mathcal{O}(d)$, we obtain a similar estimation $e_{s,p_l} \sim \mathcal{O}(d^{\frac{5}{2}})$. Since the number of stacked pseudo-copy satisfies $L = \gamma \sim \mathcal{O}(d^{-2})$, the same estimation $e_{s,p} = Le_{s,p_l} \sim \mathcal{O}(n^{-\frac{1}{2}})$ is achieved.

3.6. Proof of Theorem 2.3

Extending our results of permutation-trained networks to the random initialization scenarios imposes non-trivial challenges. Note that

the symmetry construction of the step function approximator in Eq. (4) becomes invalid with random initializations, as the error introduced by randomness can accumulate as width increases. Nevertheless, the randomly sampled parameters will become more dense upon increasing width, leading to a high probability of finding parameters that closely match the required location.

Therefore, we can first apply the UAP in the equidistant case to obtain a network f^{NN} in Eq. (2) (denoted as $f^{\mathrm{NN}}_{\mathrm{equi}}$ in the following text for distinction), which exhibits approximation power. Then, within a randomly initialized network f^{NN} in Eq. (2) (denoted as $f^{\mathrm{NN}}_{\mathrm{rand}}$) of sufficient width, we find a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ that can be regarded as randomly perturbed from $f^{\mathrm{NN}}_{\mathrm{equi}}$. If this perturbation is small enough, the subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ will also possess approximation power.

Proof of Theorem 2.3. We divide the whole proof into the following points:

(a) Approach f^* with equidistantly initialized $f^{\mathrm{NN}}_{\mathrm{equi}}$. Theorem 2.1 indicate that for any small ε and the target function f^*, there is an equidistantly initialized network $f^{\mathrm{NN}}_{\mathrm{equi}}$ in Eq. (2) with width $\tilde{n}$ and $B^{(\tilde{n})}_{\mathrm{equi}} = (b_i)_{i=1}^{\tilde{n}}$, $W^{(2\tilde{n})}_{\mathrm{equi}} = (\pm b_i)_{i=1}^{\tilde{n}}$, such that

$$\left|f^{\mathrm{NN}}_{\mathrm{equi}}(x) - f^*(x)\right| < \varepsilon/4, \quad \forall\, x \in [0,1]. \tag{27}$$

(b) Find a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ in a randomly initialized $f^{\mathrm{NN}}_{\mathrm{rand}}$ to approximate $f^{\mathrm{NN}}_{\mathrm{equi}}$. For a network with sufficiently wide $n \gg \tilde{n}$, parameters $B^{(n)}_{\mathrm{rand}} \sim \mathcal{U}[0,1]^n$ and $W^{(2n)}_{\mathrm{rand}} = (\pm p_i)_{i=1}^{n}$, $p_i \sim \mathcal{U}[0,1]$, we can find a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ with parameters

$$B^{(\tilde{n})}_{\mathrm{sub}} = \left(b_i + r_i^B\right)_{i=1}^{\tilde{n}}, \quad W^{(2\tilde{n})}_{\mathrm{sub}} = \left(\pm\left(b_i + r_i^W\right)\right)_{i=1}^{2\tilde{n}},$$

which can be viewed as randomly perturbed from $B^{(\tilde{n})}_{\mathrm{equi}}$, $W^{(2\tilde{n})}_{\mathrm{equi}}$, while the independent and identically distributed (i.i.d.) $r_i^B \sim \mathcal{U}[-\Delta r, \Delta r]^{\tilde{n}}$ and $r_i^W \sim \mathcal{U}[-\Delta r, \Delta r]^{2\tilde{n}}$ measured the perturbation, and $\Delta r > 0$ are the maximum allowable perturbation of $B^{(\tilde{n})}_{\mathrm{equi}}$ and $W^{(2\tilde{n})}_{\mathrm{equi}}$ (we impose further constraints on the parameters near the boundary of $[0,1]$ to prevent them from exceeding the range). Consequently, for sufficiently small $\Delta r < r_0$, the subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ will approach $f^{\mathrm{NN}}_{\mathrm{equi}}$ with the approximation error like

$$\left|f^{\mathrm{NN}}_{\mathrm{sub}}(x) - f^{\mathrm{NN}}_{\mathrm{equi}}(x)\right| < \varepsilon/4, \quad \forall\, x \in [0,1], \quad \text{with probability} P_{\mathrm{sub}}. \tag{28}$$

where P_{sub} denote the related possibility. This also enables $f^{\mathrm{NN}}_{\mathrm{sub}}$ the approximation power to f^* due to its continuity with respect to the parameters. Combining the results in Eq. (27)–(28), the approximation error $E^{\mathrm{sub}}_{\mathrm{rand}}$ of the subnetwork gives that, with probability P_{sub},

$$\begin{aligned} E^{\mathrm{sub}}_{\mathrm{rand}} &:= \max_{x\in[0,1]} \left|f^{\mathrm{NN}}_{\mathrm{sub}}(x) - f^*(x)\right| \\ &\le \max_{x\in[0,1]} \left|f^{\mathrm{NN}}_{\mathrm{sub}}(x) - f^{\mathrm{NN}}_{\mathrm{equi}}(x)\right| + \max_{x\in[0,1]} \left|f^{\mathrm{NN}}_{\mathrm{equi}}(x) - f^*(x)\right| < \varepsilon/2. \end{aligned} \tag{29}$$

(c) Estimate the probability P_{sub} related to the approximation power of $f^{\mathrm{NN}}_{\mathrm{sub}}$. Here we estimate the probability of finding such a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ with the required approximation error $E^{\mathrm{sub}}_{\mathrm{rand}}$ in Eq. (29). We start with the complement event $\mathcal{A}_k^c$: for a given location $\hat{b}_k \in B^{(\tilde{n})}_{\mathrm{equi}}$, there is no close enough locations in $B^{(n)}_{\mathrm{rand}}$, *i.e.*, falls in the interval $[\hat{b}_k - \Delta r, \hat{b}_k + \Delta r]$. Concretely, the probability has $\mathbb{P}[\mathcal{A}_k^c] = (1 - 2\Delta r)^n$ since the interval length is $2\Delta r$. Hence, by choosing Δr small enough such that these intervals have no overlap, *i.e.*, $\Delta r < \min\{\delta_s, 1/2\tilde{n}\}$, we apply the inclusion–exclusion principle (Feller, 1968) to write the probability of finding all locations $B^{(\tilde{n})}_{\mathrm{equi}}$ in $B^{(n)}_{\mathrm{rand}}$ as

$$\mathbb{P}\left[\bigcap_{k=1}^{\tilde{n}} \mathcal{A}_k\right] = 1 - \mathbb{P}\left[\bigcup_{k=1}^{\tilde{n}} \mathcal{A}_k^c\right] = 1 - \sum_{k=1}^{\tilde{n}} (-1)^{k+1} \binom{\tilde{n}}{k} (1 - 2k\Delta r)^n =: 1 - P'.$$

where $\binom{\tilde{n}}{k}$ is the binomial coefficient, and P' is the complement probability. This probability also holds for finding $W^{(2\tilde{n})}_{\mathrm{equi}}$ in pairwise $W^{(2n)}_{\mathrm{rand}}$. Consequently, we can find a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ within $f^{\mathrm{NN}}_{\mathrm{rand}}$ that is close enough to $f^{\mathrm{NN}}_{\mathrm{equi}}$, such that the approximation error $E^{\mathrm{sub}}_{\mathrm{rand}}$ in Eq. (29) is achieved with the probability of

$$P_{\mathrm{sub}} = \mathbb{P}\left[E^{\mathrm{sub}}_{\mathrm{rand}} < \varepsilon/2\right] = \left[1 - P'\right]^2. \tag{30}$$

For given $\tilde{n}$ and Δr, we have $P' \to 0$ as $n \to \infty$. Therefore, we choose a sufficiently large n to ensure that the probability $P_{\mathrm{sub}} \ge \sqrt{1-\delta}$.

(d) Annihilate the remaining part in the randomly initialized $f^{\mathrm{NN}}_{\mathrm{rand}}$. We follow the same discussion as in the equidistant case to eliminate the unused parameters in $f^{\mathrm{NN}}_{\mathrm{rand}}$. By applying linear reorganization and Lemma 3.1, we rewrite the remaining part as a linear function $S^r_{\mathrm{un}}(x) = \beta_r x + \eta_r$, where $x \in [0,1]$ and $0 \le \beta_r \le \Delta p$. The upper bound Δp is the largest gap between the adjacent coefficients $\{p_i\}_{i\in I_{\mathrm{un}}}$. As the network width n increases, the randomly initialized $\{p_i\}_{i\in I_{\mathrm{un}}}$ become denser, leading to $\Delta p \xrightarrow{\text{a.s.}} 0$. Therefore, we can choose a sufficiently large n to ensure $\Delta p \le \varepsilon/2$ with high probability, *i.e.*, $P_{\mathrm{un}} \ge \sqrt{1-\delta}$, where P_{un} denotes the related possibility.

Similarly to Eq. (17), we introduce an additional shift $C_r = -\eta_r$ to control the error of the intercept η_r. Therefore, the error $E^{\mathrm{un}}_{\mathrm{rand}}$ introduced by the unused part in $f^{\mathrm{NN}}_{\mathrm{rand}}$ satisfies can be estimated similarly with Eq. (17) as the following form,

$$E^{\mathrm{un}}_{\mathrm{rand}} = \max_{x\in[0,1]} \left|S^r_{\mathrm{un}}(x) + C_r\right| \le \Delta p \le \varepsilon/2, \quad \text{with probability} P_{\mathrm{un}} > \sqrt{1-\delta}. \tag{31}$$

(e) Complete the proof by combining the previous estimation. For any $\varepsilon > 0$ and $f^* \in C([0,1])$, we choose $\Delta r < \min\{r_0, \delta_s, 1/2\tilde{n}\}$ and a large n to ensure that

1. With probability P_{sub}, the subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ can approximate the f^* with the error $E^{\mathrm{sub}}_{\mathrm{rand}}$ satisfies $E^{\mathrm{sub}}_{\mathrm{rand}} < \varepsilon/2$ based on Eq. (29);
2. The probability P_{sub} satisfies $P_{\mathrm{sub}} \ge \sqrt{1-\delta}$ based on Eq. (30);
3. With probability $P_{\mathrm{un}} > \sqrt{1-\delta}$, the impact of the remaining parameters satisfies $E^{\mathrm{un}}_{\mathrm{rand}} < \varepsilon/2$ based on Eq. (31).

Hence, we can finish the proof by estimating the overall approximation error of our network $f^{\mathrm{NN}}_{\mathrm{rand}}$ to the target function f^*, which gives that with probability $1-\delta$,

$$\left|f^{\mathrm{NN}}_{\mathrm{rand}}(x) - f^*(x)\right| \le E^{\mathrm{sub}}_{\mathrm{rand}} + E^{\mathrm{un}}_{\mathrm{rand}} < \varepsilon, \quad \forall\, x \in [0,1]. \quad \square \tag{32}$$

4. Experiments

This section presents numerical evidence to support and validate the theoretical proof. An interesting observation of permutation behaviors also highlights the theoretical potential of this method.

4.1. The algorithmic implementation of permutation training

In the implementation of permutation training, guidance is crucial in finding the weights' ideal order relationship. Qiu and Suda (2020) proposed *lookahead permutation (LaPerm)* algorithm by introducing an k-times Adam-based free updating, where the learned relationship can then serve as a reference for permutation. To ensure the performance, the weights are permuted after every k epoch. The impact of k's value on convergence behavior is evaluated to be negligible (see Appendix C). Apart from the fixed permutation period k, it is also possible to adjust k to learn sufficient information for the next permutation. Refer to Appendix B for more details about the LaPerm algorithm.

4.2. Experimental settings

To validate our theoretical results, we conduct experiments on regression problems. The settings are deliberately chosen to be consistent with our proof construction. We consider a three-layer network in Eq. (2), where the first hidden layer's parameters are fixed to form

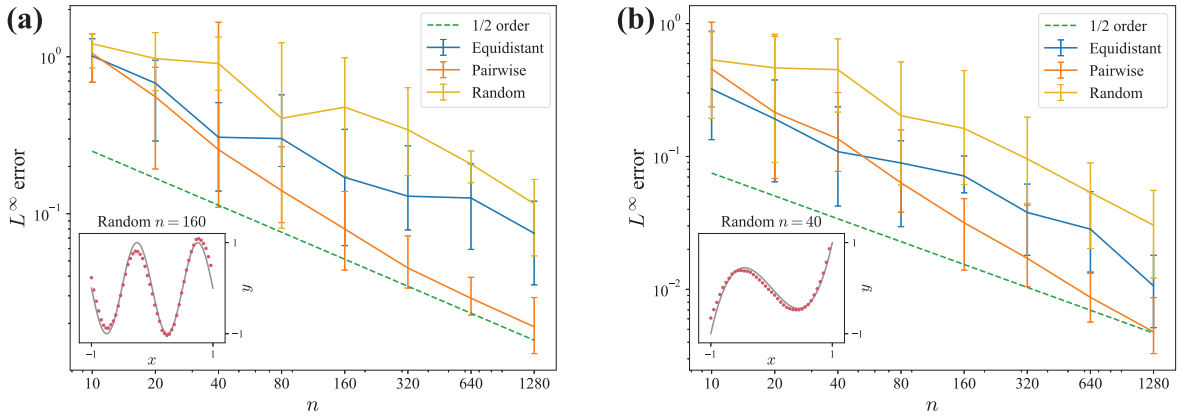


Fig. 2. Approximating one-dimensional continuous function (a): $y = -\sin(2\pi x)$ and (b): $y = \frac{1}{2}(5x^3 - 3x)$ with equidistantly, pairwise random, and randomly initialized network, where $x \in [-1, 1]$. The inset in each panel presents the target function as lines and an example of the approximation result as dots.

the basis functions $\{\phi_i^{\pm}\}_{i=1}^n$ in Eq. (1). The weights $\theta^{(2n)}$ of the second hidden layer are trained by permutation, while the scaling factors α, γ in the output layer are freely trained to reduce the required network width. All the experiments below are repeated 10 times with different random seeds, and the error bars mark the range of the maximum and minimum values. Refer to Appendix A for the detailed experimental environment and setting.

4.3. Approximating the one-dimensional continuous functions

We utilize a 1-2n-1-1 network architecture with equidistant initialization discussed in Theorem 2.1, pairwise random initializations in Theorem 2.3, and also totally random initialization $W^{(2n)} \sim \mathcal{U}[-1, 1]^{2n}$. The approximation targets are sine function $y = -\sin(2\pi x)$ and 3-order Legendre polynomial $y = \frac{1}{2}(5x^3 - 3x)$, $x \in [-1, 1]$.

The numerical result illustrated in Fig. 2 exhibits a clear convergence behavior of equidistant and pairwise random cases upon increasing n, agreeing with our theoretical proof. The clear dominance of pairwise random initialization indicates its undiscovered advantages in permutation training scenarios. Besides, the total random case also shows a certain degree of approximation power. However, in order to attain the equivalent accuracy, the total random case requires a wider network (*i.e.*, a larger n). Furthermore, the L^∞ error exhibits a $1/2$ convergence rate with respect to n. Although the theoretical estimation in Section 3 is based on L^2 norm, we indeed observe that it also holds for L^∞ error.

4.4. Approximating the multi-dimensional continuous functions

As a natural extension, we consider a two-dimensional functions $z = -\sin \pi x y$, where $(x, y) \in [-1, 1]^2$, starting with the construction of basis functions like $\phi_i^{\pm}(x)$ in Eq. (1). Recall that in the one-dimensional case, the two subsets of basis $\phi_i^{\pm}(x)$ correspond the two opposite directions along the x-axis. Therefore, at least four directions are required to represent a function defined on the xy-plane, of which two are parallel to the x-axis as $\phi_i^{\pm}(x, \cdot) = \mathrm{ReLU}(\pm(x - b_i))$ and $\phi_j^{\pm}(\cdot, y) = \mathrm{ReLU}(\pm(y - b_j))$ for y-axis, respectively. Furthermore, inspired by the lattice Boltzmann method in fluid mechanics (Chen & Doolen, 1998), we introduce another four directions as $\psi_k^{\pm\pm}(x, y) = \mathrm{ReLU}(\pm x \pm y - b_k)$. So the whole basis functions are divided into eight subsets, each corresponding to a different direction (see Fig. 3(b)). Also, the range of biases is essential since the distribution of $\psi_k^{\pm\pm}(x, y)$ must be at least $\sqrt{2}$-times wider to cover the entire domain. Here we set the biases to range in varying directions with a uniform scaling factor, providing flexibility in dealing with the unwanted coefficients.

Accordingly, we utilize a 2-8n-1-1 network architecture and follow the same setting as before (refer to Appendix A). The results depicted in Fig. 3(a) also show good approximation power. However, the $1/2$ convergence rate in previous cases cannot be attained here. We hypothesize that this is due to our preliminary eight-direction setting of the basis functions. This degeneration indicates the challenge of extending our theoretical results to higher dimensions. Further research will address this difficulty by considering more appropriate high-dimensional basis function setups. One possible approach relies on the basis construction utilized in the finite element methods (Brenner & Scott, 2008). However, adopting such a method to meet the permutation training scenario is non-trivial and requires further investigation.

Moreover, the mismatch between the existing implementations and the permutation setting poses numerical challenges to permutation training in higher dimensions. The performances are significantly affected by the algorithm implementations and initialization settings, both of which need further investigation and are beyond the scope of this work. We hope our work can inspire and motivate the development of more sophisticated implementations specific to permutation training. However, the permutation training, as a numerical algorithm, can be directly applied to high-dimensional cases, even if it requires a significantly larger network width.

Using a similar numerical setting, we can also approximate functions with three-dimensional inputs. Here we consider $f(x, y, z) = \sin 3x \cdot \cos y \cdot \sin 2z$, where $(x, y, z) \in [-1, 1]^3$. The results plotted in Fig. 4 demonstrate a certain degree of approximation power (due to the computational cost's limitation, we only conduct the experiments once). However, a degeneration convergence rate from $1/2$ to $1/6$ also indicates the theoretical limitations of the current construction.

4.5. The influence of various initialization strategies

Here we explore the effect of different initialization choices on the approximation behavior, which holds greater significance in permutation training scenarios due to the preservation of the initialized values. The case in Fig. 2(a) is utilized to apply various random initialization strategies. The results plotted in Fig. 5 show that the UAP of permutation-trained networks is not limited in the setting considered by our previous theoretical investigation. For more generalized cases, we first consider randomly initializing only $W^{(2n)}$ and $B^{(n)}$, which are labeled as Random 3 and 4, respectively. Both cases demonstrate competitive or even superior accuracy.

Next, we consider some commonly used initializations, such as Xavier's uniform initialization U_X (Glorot & Bengio, 2010), and He's normal initialization N_H (He, Zhang, Ren, & Sun, 2015). However, the implementation of U_X on Random 5 and N_H on Random 6 fails to result in convergence. This abnormal poor performance may be attributed to the mismatch of the scale in $B^{(n)}$ and the approximation

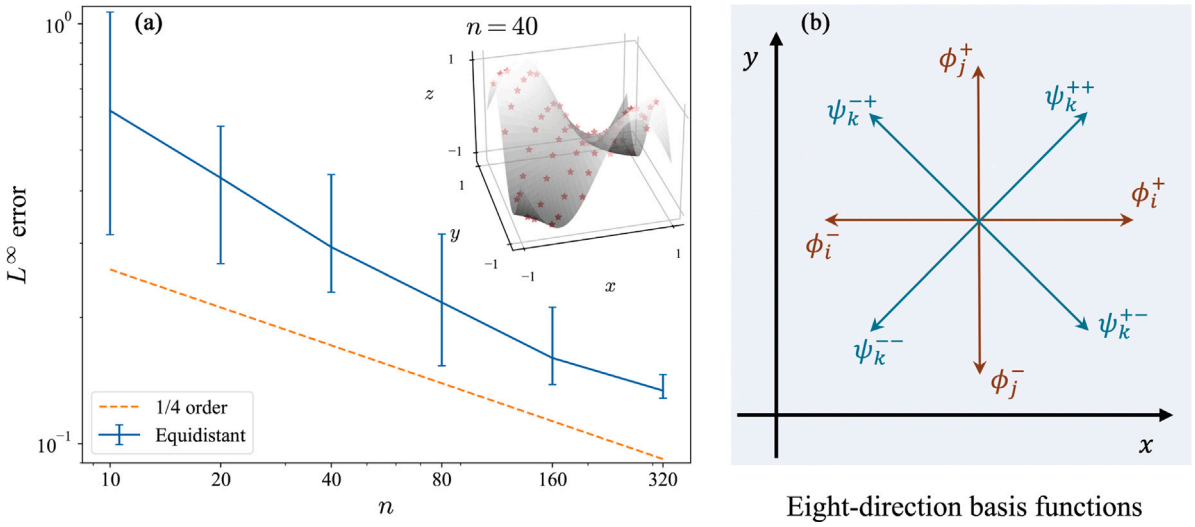


Fig. 3. (a) Approximating two-dimensional continuous function $z = -\sin \pi x y$, where $x, y \in [-1, 1] \times [-1, 1]$. The inset panel presents the target function surface and an example of the approximation result as dots. (b) The two-dimensional basis function settings.

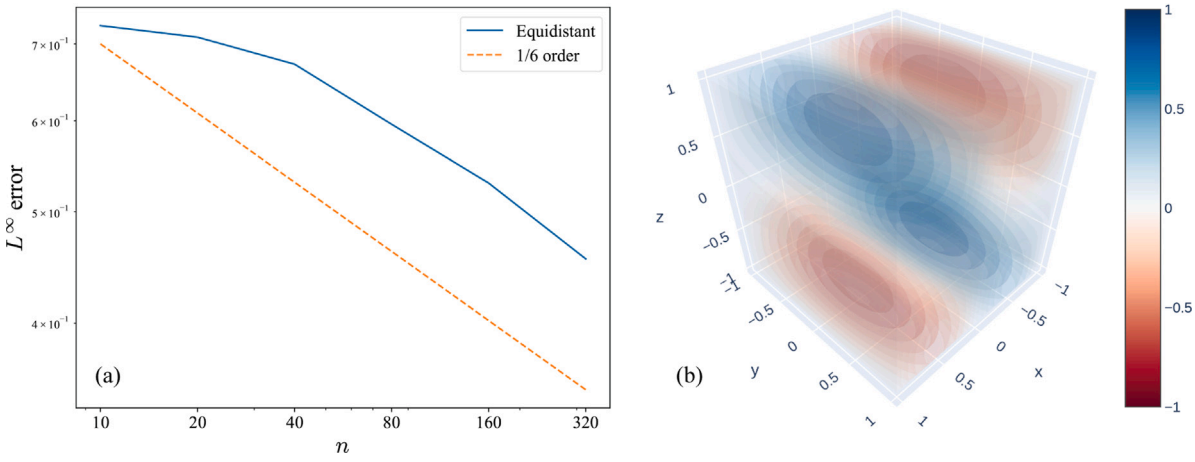


Fig. 4. Approximating three-dimensional continuous function $f(x, y, z) = \sin 3x \cdot \cos y \cdot \sin 2z$, where $(x, y, z) \in [-1, 1]^3$. (a) The convergence behavior under random seed 2022. (b) The three-dimensional illustration of the target function, where the function value $f(x, y, z)$ is plotted by the corresponding color in the color bar.

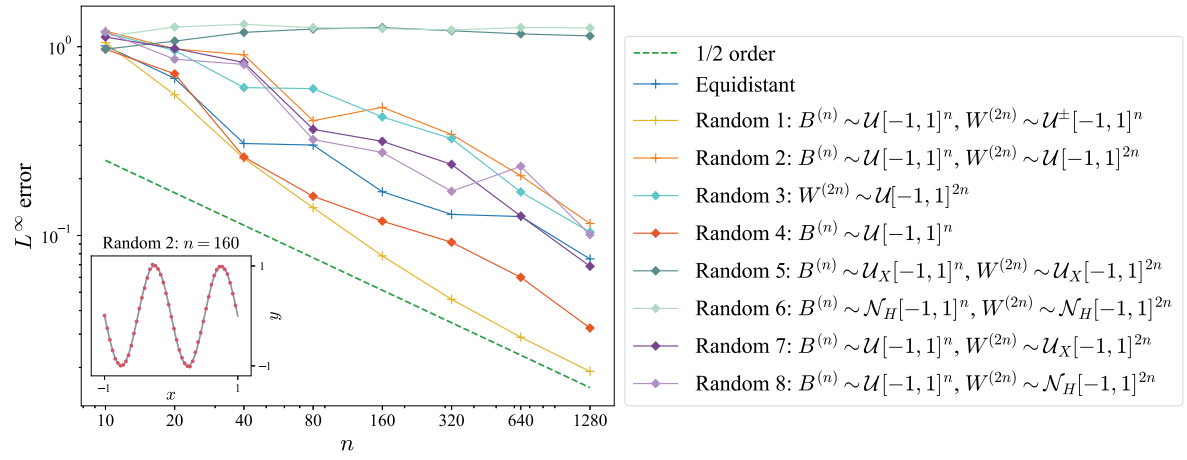


Fig. 5. The performance of different initialization strategies in approximating $y = -\sin(2\pi x)$ in $[-1, 1]$. The pairwise initialization $W^{(2n)} = (\pm p_i)_{i=1}^n$, $p_i \sim \mathcal{U}[-1, 1]$ is denoted as $W^{(2n)} \sim \mathcal{U}^{\pm}[0, 1]^n$. The error bars are omitted for conciseness. The inset panel presents the target function as lines and an example of the approximation result as dots.

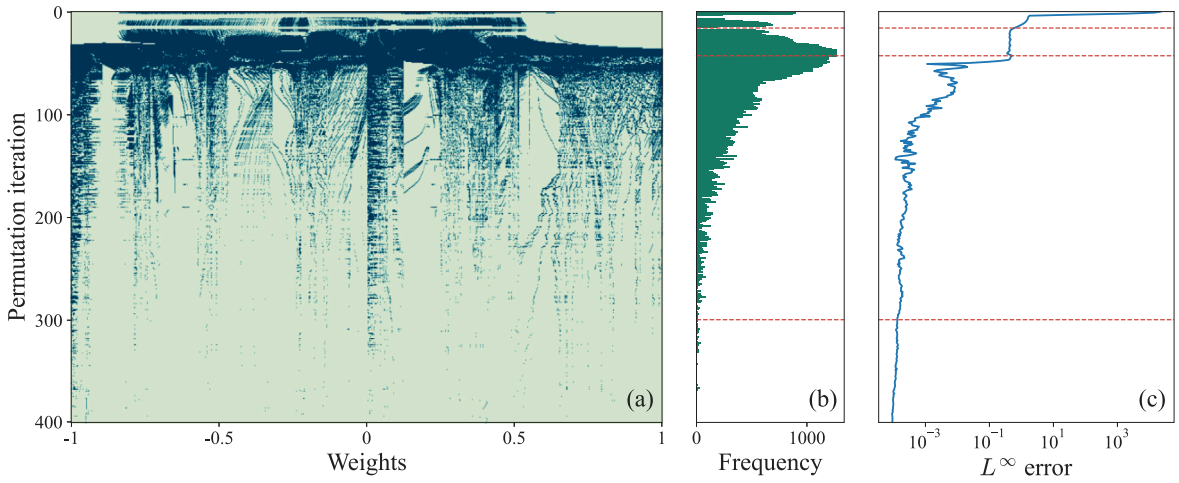


Fig. 6. The permutation behavior in the first 400 permutation iteration in approximating $y = -\sin(2\pi x)$ by equidistantly initialized network with $n = 640$. (a) The distribution of the active components (denoted by dark green color). (b) The frequency distribution illustrates the variation in the total count of active components in each permutation. (c) The corresponding loss behavior.

domain $[0, 1]$. To verify our hypothesis, we hold $B^{(n)} \sim \mathcal{U}[-1, 1]^n$, and only apply U_X and N_H on the coefficients $W^{(2n)}$ in Random 7 and 8. Consequently, both cases successfully regain the approximation ability. These surprising behaviors, especially the unexpected deterioration of the default choices, emphasizes the limited understanding of systematical characterizing the initialization suitable for permutation training scenarios.

4.6. Observation of the permutation-active patterns

This section aims to explore the theoretical potential of permutation training in describing network learning behavior. Based on the significant correlation between permutation and learning behavior, as evidenced by Qiu and Suda (2020) and our investigation, we hypothesize that the permutation-active components of the weights may play a crucial role in the training process. Therefore, by identifying and tracing the permutation-active part of weights, a novel description tool can be achieved, which also facilitates visualization and statistical analysis of the learning behavior.

As a preliminary attempt, we illustrate the permutation behavior of the coefficients $\theta^{(2n)}$ in Fig. 2(a). The components that participated in the permutation are visually highlighted in dark green in Fig. 6(a). The behaviors plotted in Fig. 6(b)–(c) clearly show that the frequency of order relationship exchange evolves synchronously with the learning process, agreeing with the observation of Qiu and Suda (2020).

Specifically, the distribution of active components shows significant patterns, which are classified into four stages (marked by red dash lines in Fig. 6). The loss declines sharply in the initial stage, while only the components with medium value are permuted. Once loss reaches a plateau in the second stage, more components are involved in permutation, evidencing the role of permutation in propelling the training. As loss starts to decline again, the permutation frequency correspondingly diminishes. Interestingly, the slower loss decrease gives rise to a ribbon-like pattern, akin to the reported localized permutations (plotted in Fig. 14 of the appendix in Qiu and Suda (2020)), and possibly due to slow updates failing to trigger a permutation. This observation may support the existence of inherent low-dimensional structures within the permutation training dynamics, potentially linked to mathematical depiction of permutation groups, such as cycle decomposition (Cameron, 1999) and Fourier bases for permutation (Huang, Guestrin, & Guibas, 2009). Finally, the permutation's saturation aligns with the stationary state of loss convergence. We believe these inspiring phenomena deserve further exploration, as they hold the potential to provide novel insights regarding the learning behavior of networks.

5. Discussion and future work

Despite the exploratory nature as the first theoretical work (to our knowledge), our findings suggest the valuable potential of permutation training. Some intriguing questions are also raised for future research.

5.1. Generalizing to other scenarios

Although we mainly focus on basic settings, the proof idea exhibits generalizability. Extending to networks equipped with leaky-ReLU can be straightforward (refer to Appendix D for numerical evidence). Our proof also enables implementations within other designs, such as networks with deeper architectures or sparse initializations (see Appendix E for detailed discussion).

However, extending our theoretical results to the high-dimensional scenario still faces challenges. One of the primary obstacles is constructing multi-dimensional basis functions that are suitable for the permutation training scenario. A reasonable approach relies on the construction in the finite element methods (Brenner & Scott, 2008). We plan to address this problem in future work. As for the numerical evidence of high-dimensional scenarios, Qiu and Suda (2020) have examined the classification problem using VGG-based networks on the CIFAR-10 dataset, while our experiments in regression task have shown approximation behavior for two and three-dimensional inputs (see Section 4.4).

5.2. Permutation training as a theoretical tool

Our observation in Section 4.6 indicates the theoretical potential of permutation training, as it corresponds well with the training process and has systematical mathematical descriptions. Specifically, the patterns observed in Fig. 6 can intuitively lead to certain weight categorization strategies, potentially benefit consolidating the crucial weights for previous tasks (Maltoni & Lomonaco, 2019), or pruning to find the ideal subnetwork in the lottery ticket hypothesis (Frankle & Carbin, 2019). Additionally, the existing permutation training algorithm shares

the same form as weight projection methods in continual learning (Zeng et al., 2019), as it can be viewed as applying an order-preserving projection from the free training results to the initial weight value.

5.3. The algorithmic implementation

This work is expected to facilitate the applications of permutation training. However, some practical issues still exist and deserve further investigation. As a preliminary attempt, the existing permutation training algorithm LaPerm guides the permutation by inner loops, thus incurring undesirable external computation costs. However, employing more advanced and efficient search approaches, such as the learn-to-rank formalism (Cao, Qin, Liu, Tsai, & Li, 2007), or permutation search algorithms in the study of LMC (Ainsworth et al., 2023; Jordan et al., 2023), the benefits of permutation training will be actualized in practice. Importantly, our proof does not rely on any specific algorithmic implementations. Additionally, the incompatible initialization issue plotted in Fig. 2(b) emphasizes the need for developing more effective initializations as well as investigating the criterion of UAP-compatible initializations.

6. Conclusion

As a novel method, permutation training exhibits unique properties and practical potential. To verify its efficacy, we prove the UAP of permutation-trained networks with random initialization for one-dimensional continuous functions. The proof is generalized from the equidistant scenario, where the key idea involves a four-pair construction of step function approximators in Fig. 1, along with a processing method to eliminate the impact of the remaining parameters. Our numerical experiments not only confirm the theoretical results through Fig. 2(a), but also validate the prevalence of the UAP of permutation-trained networks in various initializations in Fig. 2(b). The discovery that commonly used initializations fail to achieve UAP also raises an intriguing question about the systematic characterization of initializations that satisfy UAP. Our observation in Fig. 6 suggests that permutation training could be a novel tool to describe the network learning behavior.

Overall, we believe that the UAP of permutation-trained networks reveals the profound, yet untapped insights into how the weight encodes the learned information, highlighting the importance of further theoretical and practical exploration.

CRediT authorship contribution statement

Yongqiang Cai: Writing – review & editing, Writing – original draft, Visualization, Validation, Project administration, Formal analysis, Conceptualization. **Gaohang Chen:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Zhonghua Qiao:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the CAS AMSS-PolyU Joint Laboratory of Applied Mathematics, Hong Kong. The work of Y. Cai is supported by the National Natural Science Foundation of China under Grant 12201053 and 12171043. The work of Z. Qiao is supported by the Hong Kong Research Grants Council RFS grant RFS2021-5S03 and GRF grant 15302122, and the Hong Kong Polytechnic University grant 4-ZZLS.

Appendix A. The experimental setting

To establish the convergence property upon increasing network width, we sample the training points randomly and uniformly in $[-1, 1]$, along with equidistantly distributed test points. The maximum training epoch is sufficiently large to ensure reaching the stable state. For the multi-dimensional case, we set the basis functions at a larger domain than the functions to ensure accuracy near the boundary. The scale is measured by T_b, which means the biases are in $[-1-T_b, 1+T_b]$ in each dimension. See Table A.1 for detailed choice.

The experiments are conducted in NVIDIA A100 Tensor Core GPU. However, our code is hardware-friendly since each case only consumes approximately 2 GB of memory. The code uses *torch.multiprocessing* in PyTorch 2.0.0 with ten different random seeds, namely $2022, 3022, \dots, 12022$. Additionally, the training data of each case is sampled under the random seed 2022 to ensure that they are comparable.

Appendix B. Adam-based LaPerm algorithm

Here we briefly introduce the Adam-based LaPerm algorithm with a fixed permutation period k in Qiu and Suda (2020). The pseudocode is shown in Algorithm 1. This algorithm rearranges the initial weights W guided by the order relationship of θ_t, so the trained weights will hold the initial value. Therefore, it can be regarded as a permutation of the initial weights W.

Algorithm 1 Adam-based LaPerm algorithm.

Require: Loss function $\mathcal{L}$, training set D_T, maximum training epoch M_e
Require: Inner optimizer Adam, permutation period k, initial weights W
  $\theta_0 = W$ // Initialize the weights
  for $t = 1, 2, \dots, M_e$ **do**
    $\theta_t \leftarrow \text{Adam}(\mathcal{L}, \theta_{t-1}, D_T)$ // Free training by Adam
    if k divides$t$ **then**
      $\theta_t \leftarrow \tau_t(W)$ // Apply the permutation
    end if
  end for

Appendix C. The impact of permutation period on convergence behavior

As a hyperparameter, the choice of permutation period k during the implementation of LaPerm algorithms has the possibility to affect the convergence behavior. The correlation between the value of k and the final accuracy is reported to be unambiguous (refer to Fig. 6 in Qiu and Suda (2020)). Generally, a larger k is associated with slightly higher accuracy of single permutation training result, thus in our experiments, the weights are permuted after each k epoch. Fig. C.7 evaluates the impact of k's value on convergence behavior, whose results suggest that this effect remains negligible.

Table A.1
Hyperparameters setting.

Hyperparameters	1D	2D	3D
Architectures	1-2n-1-1	2-8n-1-1	3-26n-1-1
k	5	5	20
Batch size	8	128	640
# training points	1600		51200
# test points	400		12800
T_b	0		0.75
n		$\{10, 20, 40, 80, 160, 320\}$	
# epoch		6400	
Learning rate (LR)		1e-3	
Multiplicative factor of LR decay		0.998	
Multiplicative factor of k increase		$\sqrt[10]{1.002}$	

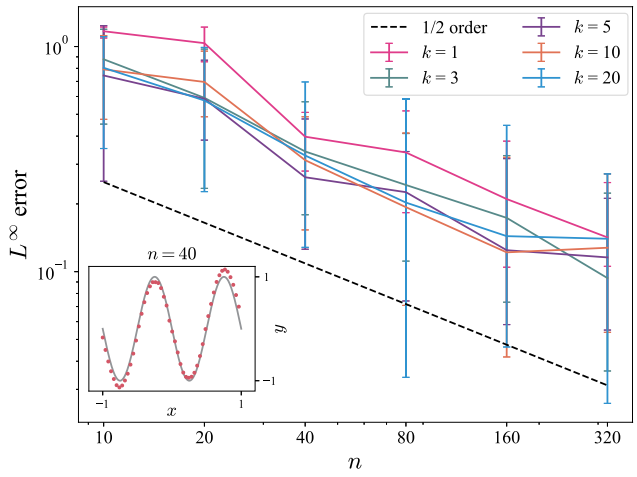


Fig. C.7. Approximating one-dimensional continuous function $y = a_0 + a_1\sin(\pi x) + a_2\cos(2\pi x) + a_3\sin(3\pi x)$ with equidistantly initialized network, where $x \in [-1, 1]$, and the value of permutation period $k = 1, 3, 5, 10, 20$, respectively. The inset in each panel presents the target function as lines and an example of the approximation result as dots.

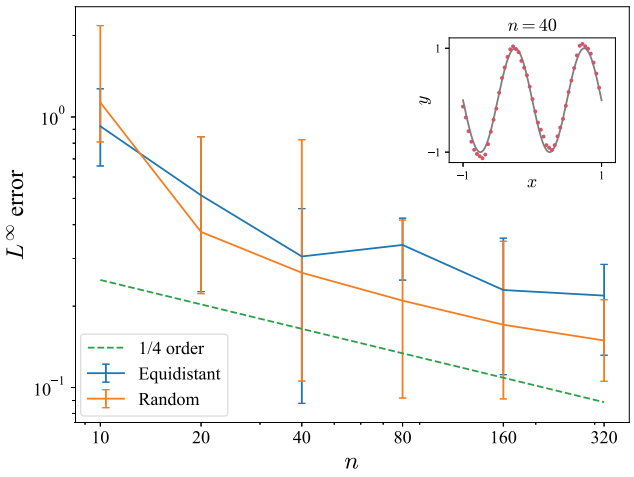


Fig. D.8. Approximating one-dimensional continuous function $y = -\sin(2\pi x)$ with equidistantly initialized network equipped with leaky-ReLU, where $x \in [-1, 1]$. The inset in each panel presents the target function as lines and an example of the approximation result as dots.

Appendix D. Generalize to the networks equipped with leaky-ReLU

Extending our outcomes to leaky-ReLU is expected. This is because of the two crucial techniques deployed in our proof: constructing the step function approximators and eliminating the unused parameters, both can be applied to the leaky-ReLU.

Since our two-direction setting of basis function $\phi^{\pm}$ in Eq. (1) can impart leaky-ReLU with symmetry equivalent to ReLU, it is feasible to construct a similar step function approximator by rederiving the relevant coefficient p_i, q_i. Furthermore, the existing eliminating method can be directly employed since a pair of leaky-ReLU basis functions can be constructed into linear functions for further processing.

As an initial attempt, we numerically examine the leaky-ReLU networks by only changing the activation function in the case of Fig. 2(a). The results plotted in Fig. D.8 exhibit the approximation power of leaky-ReLU networks. Unlike the ReLU cases, the random initialization outperforms the equidistant initialization. However, the $1/2$ convergence rate in previous ReLU cases cannot be attained here, probably because the proof based on leaky-ReLU may result in different constants, leading to potential discrepancies in details when compared with the ReLU-based conclusions.

Appendix E. Extending to the networks with deeper architecture or sparse initializations

This paper follows the conventional setting of free training UAP study and primarily focuses on shallow fully connected networks. This is based on the premise that the conclusions can be readily extended to scenarios involving deeper networks. As for deep networks, the permutation training setting seems to inhibit a direct extension. However, our linear reorganization in Eq. (12) enables construct an identity function $y = x$ using a pair of basis functions $y = p_n\phi_1^+(x) + q_n\phi_1^-(x)$, where $b_1 = 0$, $p_n = 1$, $q_n = -1$. This process enables us to utilize identity functions within subsequent layers. Consequently, the deep networks scenario parallels the cases discussed in this paper, which allows us to realize UAP within deep networks.

Additionally, this paper's conclusions could be extended to sparse initialization scenarios, an area where permutation training shows significant advantages (see Fig. 6(e) in Qiu and Suda (2020)). On an intuitive level, our proof can explain these advantages, as a primary challenge of permutation training is managing unused basis functions. However, in sparse situations, we can conveniently deal with this problem by assigning the remaining basis functions zero coefficients, thereby facilitating the proof process.

Appendix F. A toy example of our step-matching and constant-matching construction

To illustrate the key idea of our construction, we consider a toy example of approaching a step function $f_s(x) = 0.8\,\chi(x - 0.4)$ for $x \in [0, 1]$ by a network in Eq. (2) with $n = 11$ and equidistantly initialized $\{b_k\}_{k=1}^{11} = \{0, 0.1, \dots, 1\}$. This setting is considered after applying Lemma 2.1 with $\Delta h = 0.8$.

Following our step-matching construction, we can choose the basis functions with $\{b_k\}_{k=1}^{4} = \{0.1, 0.3, 0.6, 0.8\}$ to be step-matching in Eq. (6), leading to a step function approximator f_s^{NN}. The height of f_s^{NN} is given by $h = 0.4$. Therefore, the target step function f_s can be approximated by f_s^{NN} along with shifting $\Delta h/2 = 0.4$ and scaling $\Delta h/h = 2$ with the following error estimation:

$$\left|\left[\frac{\Delta h}{h}(x)\, f_s^{\mathrm{NN}} + \frac{h}{2}\right] - f_s(x)\right| \le 0.8 = \Delta h, \quad x \in [0, 1].$$

The target step function f_s and its approximator f_s^{NN} are plotted in Fig. F.9.

Additionally, we can consider the constant function approximator f_c^{NN} following our constant-matching construction in Eq. (10). We choose the basis functions located at $\{0.2, 0.4, 0.7, 0.9\}$ to be constant-matching, leading to a constant function approximator $f_c^{\mathrm{NN}} = h/2$. The height of f_c^{NN} is given by $h/2 = 0.4$. From the shape of f_c^{NN} plotted in Fig. F.9, we can see that it can approximate a constant function without any error.

Appendix G. A toy example of Lemma 3.1

To illustrate the processing method of Lemma 3.1, we consider a given sequence

$$3,\ 1,\ 4,\ -1,\ 5,\ 9,$$

with the largest gap $\Delta c = 4$. The aim is to choose $m_i \in \{-1, 1\}$ to reduce the absolute value of the result

$$S = 3 \cdot m_1 + 1 \cdot m_2 + 4 \cdot m_3 - 1 \cdot m_4 + 5 \cdot m_5 + 9 \cdot m_6.$$

This can be achieved by the following steps:

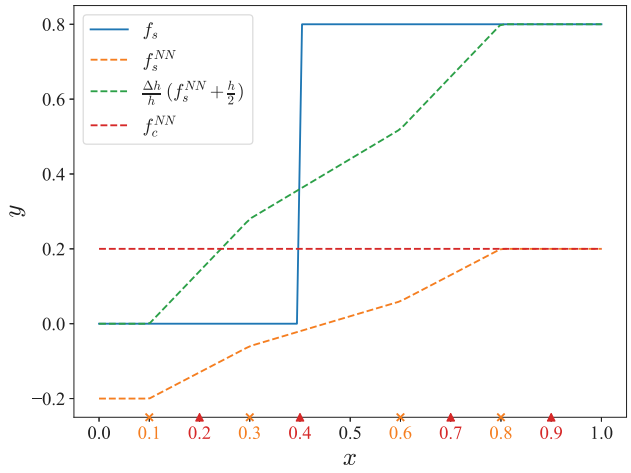


Fig. F.9. Approximating a step function $f_s(x) = 0.8\,\chi(x - 0.4)$ and a constant function $f_c(x) = 0.2$ in $x \in [0, 1]$ with the step-function approximator f_s^{NN} and the constant function approximator f_c^{NN}, respectively. The target functions are plotted as lines, and the approximation results are shown as dashed lines. The locations of the basis functions used for f_s^{NN} and f_c^{NN} are marked with "×" and "▲" on the x-axis, respectively.

1. Rearrange the sequence into 9, 5, 4, 3, 1, −1;
2. Compute the gap between every two adjacent elements as 4, 1, 2;
3. Reorganize the gap sequence into an alternating series and compute the result as $S = 4 - 2 + 1 = 3$;
4. Transfer the choice of $m_i \in \{-1, 1\}$ to the original sequence as

$$S = (9 - 5) - [1 - (-1)] + (4 - 3) = 9 - 5 - 1 - 1 + 4 - 3 = 3 < 4 = \Delta c.$$

Therefore, the result $S = 3$ can be achieved by choosing $m_1 = -1, m_2 = -1, m_3 = 1, m_4 = 1, m_5 = -1, m_6 = 1$.

Data availability

The code and data accompanying this manuscript are publicly available on GitHub at https://github.com/DanclaChen/PermutationTraining.

References

Ainsworth, S., Hayase, J., & Srinivasa, S. (2023). Git re-basin: Merging models modulo permutation symmetries. In *The eleventh international conference on learning representations*.

Brenner, S. C., & Scott, L. R. (2008). *The mathematical theory of finite element methods*. Springer.

Cai, Y. (2023). Achieve the minimum width of neural networks for universal approximation. In *The eleventh international conference on learning representations*.

Cai, Z., Chen, J., & Liu, M. (2021). Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation. *Journal of Computational Physics*, *443*, Article 110514.

Cameron, P. J. (1999). *Permutation groups*: *vol. 45*, Cambridge University Press.

Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on machine learning* (pp. 129–136).

Castro, J. L., Mantas, C. J., & Benıtez, J. (2000). Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks*, *13*, 561–563.

Chen, S., & Doolen, G. D. (1998). Lattice boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, *30*, 329–364.

Cohen, T., & Welling, M. (2016). Group equivariant convolutional networks. In *International conference on machine learning* (pp. 2990–2999). PMLR.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*, 303–314.

Entezari, R., Sedghi, H., Saukh, O., & Neyshabur, B. (2021). The role of permutation invariance in linear mode connectivity of neural networks. In *International conference on learning representations*.

Fan, F., Xiong, J., & Wang, G. (2020). Universal approximation with quadratic deep networks. *Neural Networks*, *124*, 383–392.

Feldmann, J., Youngblood, N., Karpov, M., Gehring, H., Li, X., Stappers, M., et al. (2021). Parallel convolutional processing using an integrated photonic tensor core. *Nature*, *589*, 52–58.

Feller, W. (1968). *An introduction to probability theory and its applications, volume i* (3rd ed.). John Wiley & Sons.

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the international conference on learning representations*.

Frankle, J., Dziugaite, G. K., Roy, D., & Carbin, M. (2020). Linear mode connectivity and the lottery ticket hypothesis. In *International conference on machine learning* (pp. 3259–3269). PMLR.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR workshop and conference proceedings* (pp. 249–256).

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034).

Hopcroft, J. E., Ullman, J. D., & Aho, A. V. (1983). *Data structures and algorithms*: *vol. 175*, MA, USA: Addison-wesley Boston.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*, 359–366.

Huang, J., Guestrin, C., & Guibas, L. (2009). Fourier theoretic probabilistic inference over permutations. *Journal of Machine Learning Research*, *10*.

Hungerford, T. W. (2012). *Algebra*: *vol. 73*, Springer Science & Business Media.

Jordan, K., Sedghi, H., Saukh, O., Entezari, R., & Neyshabur, B. (2023). REPAIR: Renormalizing permuted activations for interpolation repair. In *The eleventh international conference on representation learning*.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the international conference on learning representations*.

Kosuge, A., Hamada, M., & Kuroda, T. (2021). A 16 nj/classification fpga-based wired-logic dnn accelerator using fixed-weight non-linear neural net. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *11*, 751–761.

Kosuge, A., Hsu, Y. C., Hamada, M., & Kuroda, T. (2021). A 0.61-μj/frame pipelined wired-logic dnn processor in 16-nm fpga using convolutional non-linear neural network. *IEEE Open Journal of Circuits and Systems*, *3*, 4–14.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., & Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th international conference on machine learning* (pp. 3744–3753). PMLR.

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

Lu, L., Jin, P., Pang, G., Zhang, Z., & Karniadakis, G. E. (2021). Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, *3*, 218–229.

Lu, Z., Pu, H., Wang, F., Hu, Z., & Wang, L. (2017). The expressive power of neural networks: A view from the width.

Maltoni, D., & Lomonaco, V. (2019). Continuous learning in single-incremental-task scenarios. *Neural Networks*, *116*, 56–73.

Maron, H., Ben-Hamu, H., Shamir, N., & Lipman, Y. (2019). Invariant and equivariant graph networks. In *International conference on learning representations*.

Nugent, A. (2005). Physical neural network design incorporating nanotechnology. In *US Patent 6 889* (p. 216).

Qiu, Y., & Suda, R. (2020). Train-by-reconnect: Decoupling locations of weights from their values. *Advances in Neural Information Processing Systems*, *33*, 20952–20964.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707.

Rudin, W. (1976). *Principles of mathematical analysis* (3d ed.). New York: McGraw-Hill.

Satorras, V. G., Hoogeboom, E., & Welling, M. (2021). E (n) equivariant graph neural networks. In *International conference on machine learning* (pp. 9323–9332). PMLR.

Scabini, L., De Baets, B., & Bruno, O. M. (2024). Improving deep neural network random initialization through neuronal rewiring. *Neurocomputing*, *599*, 128130.

Shen, Z., Yang, H., & Zhang, S. (2022). Optimal approximation rate of relu networks in terms of width and depth. *Journal de Mathématiques Pures et Appliquées*, *157*, 101–135.

Stein, E. M., & Shakarchi, R. (2009). *Real analysis: measure theory, integration, and Hilbert spaces*. Princeton University Press.

Stone, M. H. (1948). The generalized weierstrass approximation theorem. *Mathematics Magazine*, *21*, 237–254.

Telgarsky, M. (2016). Benefits of depth in neural networks. In *Conference on learning theory* (pp. 1517–1539). PMLR.

Wang, M. X., & Qu, Y. (2022). Approximation capabilities of neural networks on unbounded domains. *Neural Networks*, *145*, 56–67.

Yarotsky, D. (2017). Error bounds for approximations with deep relu networks. *Neural Networks*, *94*, 103–114.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). Deep sets. In *Advances in neural information processing systems*.

Zeng, G., Chen, Y., Cui, B., & Yu, S. (2019). Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, *1*, 364–372.