Journal of Computational Mathematics
Vol.xx, No.x, 2023, 1–29.

http://www.global-sci.org/jcm
doi:10.4208/jcm.2307-m2022-0233

TENSOR NEURAL NETWORK AND ITS NUMERICAL INTEGRATION*

Yifan Wang
LSEC, NCMIS, Institute of Computational Mathematics, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China
Email: wangyifan@lsec.cc.ac.cn
Pengzhan Jin
School of Mathematical Sciences, Peking University, Beijing 100871, China
Email: jpz@pku.edu.cn
Hehu Xie[1)]
LSEC, NCMIS, Institute of Computational Mathematics, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China
Email: hhxie@lsec.cc.ac.cn

Abstract

In this paper, we introduce a type of tensor neural network. For the first time, we propose its numerical integration scheme and prove the computational complexity to be the polynomial scale of the dimension. Based on the tensor product structure, we develop an efficient numerical integration method by using fixed quadrature points for the functions of the tensor neural network. The corresponding machine learning method is also introduced for solving high-dimensional problems. Some numerical examples are also provided to validate the theoretical results and the numerical algorithm.

Mathematics subject classification: 65N30, 65N25, 65L15, 65B99.
Key words: Tensor neural network, Numerical integration, Fixed quadrature points, Machine learning, High-dimensional eigenvalue problem.

1. Introduction

Partial differential equations (PDEs) appear in many scientific and industrial applications since they can describe physical and engineering phenomena or processes. So far, many types of numerical methods have been developed such as the finite difference method, finite element method, and spectral method for solving PDEs in three spatial dimensions plus the temporal dimension. But there exist many high-dimensional PDEs such as many-body Schrödinger, Boltzmann equations, Fokker-Planck equations, and stochastic PDEs (SPDEs), which are almost impossible to be solved using traditional numerical methods. Recently, many numerical methods have been proposed based on machine learning to solve the high-dimensional PDEs [2, 6, 7, 14, 25, 28, 31, 37, 38]. Among these machine learning methods, neural network-based methods attract more and more attention. Neural networks can be used to build approximations of the exact solutions of PDEs by machine learning methods. The reason is that

* Received October 19, 2022 / Revised version received January 17, 2023 / Accepted July 26, 2023 / Published online December 4, 2023 /
1) Corresponding author

neural networks can approximate any function given enough parameters. This type of method provides a possible way to solve many useful high-dimensional PDEs from physics, chemistry, biology, engineering, and so on.

Due to its universal approximation property, the fully-connected neural network (FNN) is the most widely used architecture to build the functions for solving high-dimensional PDEs. There are several types of FNN-based methods such as well-known the deep Ritz [7], deep Galerkin method [37], PINN [31], and weak adversarial networks [38] for solving high-dimensional PDEs by designing different loss functions. Among these methods, the loss functions always include computing high-dimensional integration for the functions defined by FNN. For example, the loss functions of the deep Ritz method require computing the integrations on the high-dimensional domain for the functions which is constructed by FNN. Direct numerical integration for the high-dimensional functions also meets the "curse of dimensionality". Always, the Monte-Carlo method is adopted to do the high-dimensional integration with some types of sampling methods [7, 15]. Due to the low convergence rate of the Monte-Carlo method, the solutions obtained by the FNN-based numerical methods are difficult to obtain high accuracy and stable convergence process. In other words, the Monte-Carlo method decreases computational work in each forward propagation by decreasing the simulation efficiency and stability of the FNN-based numerical methods for solving high-dimensional PDEs.

The CANDECOMP/PARAFAC (CP) tensor decomposition builds a low-rank approximation method and is a widely used way to cope with the curse of dimensionality. The CP method decomposes a tensor as a sum of rank-one tensors which can be considered as the higher-order extensions of the singular value decomposition (SVD) for the matrices. This means the SVD idea can be generalized to the decomposition of the high-dimensional Hilbert space into the tensor product of several Hilbert spaces. The tensor product decomposition has been used to establish low-rank approximations of operators and functions [5, 13, 19, 32]. If we use the low-rank approximation to do the numerical integration, the computational complexity can avoid the exponential dependence on the dimension in some cases [4, 28]. Inspired by CP decomposition, this paper focuses on a special low-rank neural networks structure and its numerical integration. It is worth mentioning that although CP decomposition should be useful to obtain a low-rank approximation, there is no known general result to give the relationship between the rank (hyperparameter p in this paper) and error bounds. For more details, please refer to [17, 23] and numerical investigations [5].

This paper aims to propose a type of tensor neural network (TNN) to build the trial functions for solving high-dimensional PDEs. The TNN is a function being designed by the tensor product operations on the neural networks or by low-rank approximations of FNNs. An important advantage is that we do not need to use Monte-Carlo method to do the integration for the functions which is constructed by TNN. This is the main motivation to design the TNN for high-dimensional PDEs in this paper. We will show, the computational work for the integration of the functions by TNN is only a polynomial scale of the dimension, which means the TNN overcomes the "curse of dimensionality" in some sense for solving high-dimensional PDEs.

An outline of the paper goes as follows. In Section 2, we introduce the way to build TNN. The numerical integration method for the functions constructed by TNN is designed in Section 3. Section 4 is devoted to proposing the TNN-based machine learning method for solving the high-dimensional eigenvalue problem with the numerical integration method. Some numerical examples are provided in Section 5 to show the validity and efficiency of the proposed numerical methods in this paper. Some concluding remarks are given in the last section.

2. Tensor Neural Network

2.1. Architecture of tensor neural network

In this section, we introduce the TNN and its approximation property. Without loss of generality, we first design the general TNN architecture with K-dimensional output to accommodate more general computational aims. Then we consider the one-dimensional output TNN architecture which is our primary focus in this paper. Of course, it is easy to know that the K-dimensional output TNN can also be built with K one-dimensional output TNN. The approximation property for the one-dimensional TNN, which will be given in this section, can be directly extended to K-dimensional output TNN.

The architecture of TNN is similar to MIONet, just by setting the Banach spaces to Euclidean spaces, more details about MIONet can be found in [19]. MIONet mainly discusses the approximation of multiple-input continuous operators by low-rank neural network structures and investigates the function approximation under C-norm. The inputs of MIONet are the vectors that denote the coefficients of projections of functions in infinite-dimensional Banach spaces onto the concerned finite-dimensional subspace. While TNN considers solving high-dimensional PDEs and pays more attention to the high-dimensional integration and the approximation to functions in Sobolev space by low-rank neural network structures in H^m-norm. Different from MIONet, the inputs of TNN are coordinates in the high-dimensional Euclidean space. The tensor product structure of TNN can lead to high precision and high efficiency in calculating the numerical integrations in the loss function derived from the variational principle. The most important contribution of this paper is to reveal that we can do the highly accurate and efficient numerical integrations of TNN for solving high-dimensional PDEs with TNN. Furthermore, this paper also shows that the Monte-Carlo or stochastic sampling process is not necessary for machine learning. More specifically, we first construct d subnetworks, where each subnetwork is a continuous mapping from a bounded closed set $\Omega_i \subset \mathbb{R}$ to $\mathbb{R}^p$. The i-th subnetwork can be expressed as

$$\mathbf{\Phi}_i(x_i;\theta_i) = \big(\phi_{i,1}(x_i;\theta_i), \phi_{i,2}(x_i;\theta_i), \cdots, \phi_{i,p}(x_i;\theta_i)\big)^T, \quad i = 1, \ldots, d, \tag{2.1}$$

where x_i denotes the one-dimensional input, θ_i denotes the parameters of the i-th subnetwork, typically the weights and biases. The number of layers and neurons in each layer, the selections of activation functions, and other hyperparameters can be different in different subnetworks. In this paper, we simply use FNN architectures for each subnetwork. It is worth mentioning that, in addition to FNN, other reasonable architecture can be used as long as it can approximate any mapping from $\Omega_i \subset \mathbb{R}$ to $\mathbb{R}^p$ in some sense. The only thing to be guaranteed is that the output dimensions of these subnetworks should be equal. After building each subnetwork, we combine the output layers of each subnetwork to obtain TNN architecture by the following mapping from $\mathbb{R}^d$ to $\mathbb{R}^K$:

$$\mathbf{\Psi}(x;\theta) = W \cdot \big(\mathbf{\Phi}_1(x_1;\theta_1) \odot \mathbf{\Phi}_2(x_2;\theta_2) \odot \cdots \odot \mathbf{\Phi}_d(x_d;\theta_d)\big), \tag{2.2}$$

where $\odot$ is the Hadamard product (i.e. element-wise product), $\mathbf{\Psi}$ denotes a K-dimensional output function which is defined as $\mathbf{\Psi}(x;\theta) = (\Psi_1(x;\theta), \cdots, \Psi_K(x;\theta))^T$, the matrix $W \in \mathbb{R}^{K\times p}$ and $x = (x_1, \cdots, x_d) \in \Omega_1 \times \cdots \times \Omega_d$. In the following part of this paper, we set $\Omega = \Omega_1 \times \cdots \times \Omega_d$. Here $\theta = \{\theta_1, \cdots, \theta_d, W\}$ denote trainable parameters. Fig. 2.1 shows the architecture of K-dimensional output TNN.

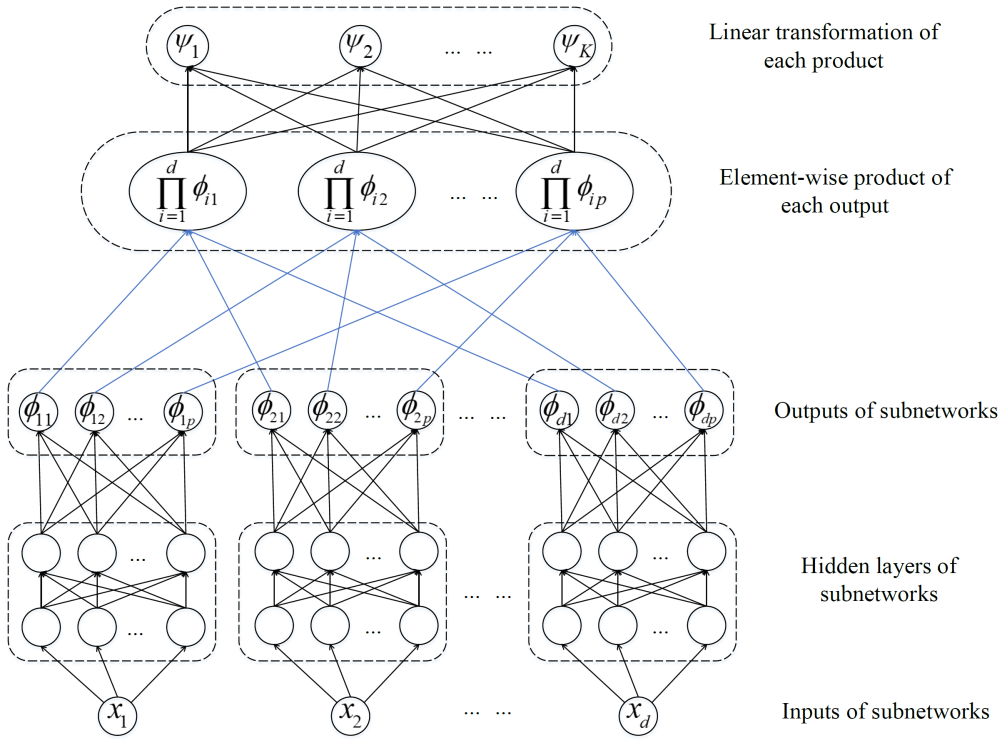

Fig. 2.1. The architecture of K-dimensional output TNN. Black arrows mean linear transformation (or affine transformation). Each ending node of blue arrows is obtained by taking the scalar multiplication of all starting nodes of blue arrows that end in this ending node.

The one-dimensional output TNN (i.e. $K = 1$) is always enough for solving normal high-dimensional PDEs. When $K = 1$, the matrix W appears in (2.2) degenerates to a row vector, its members only play the role to scale the components of vectors obtained by the Hadamard product. This effect can also be achieved by scaling the parameters of the output layers of the concerned subnetworks. Therefore, to reduce the number of parameters, we set the matrix W to be unity and define the one-dimensional TNN as follows:

$$\Psi(x;\theta) = \sum_{j=1}^{p} \phi_{1,j}(x_1;\theta_1)\phi_{2,j}(x_2;\theta_2)\cdots\phi_{d,j}(x_d;\theta_d) = \sum_{j=1}^{p}\prod_{i=1}^{d}\phi_{i,j}(x_i;\theta_i), \tag{2.3}$$

where $\theta = \{\theta_1, \cdots, \theta_d\}$ denotes all parameters of the whole architecture. Fig. 2.2 shows the corresponding architecture of one-dimensional output TNN. For simplicity, TNN refers to the one-dimensional TNN hereafter in this paper.

Since there exists the isomorphism relation between $H^m(\Omega_1 \times \cdots \times \Omega_d)$ and the tensor product space $H^m(\Omega_1) \otimes \cdots \otimes L^2(\Omega_d)$, the process of approximating the function $f(x) \in H^m(\Omega_1 \times \cdots \times \Omega_d)$ with the TNN defined by (2.3) can be regarded as searching for a CP decomposition structure to approximate $f(x)$ in the space $H^m(\Omega_1) \otimes \cdots \otimes H^m(\Omega_d)$ with the rank being not greater than p. Due to the low-rank structure, we will find that the polynomial compound acting on the TNN and its derivatives can be integrated with small-scale computational work.

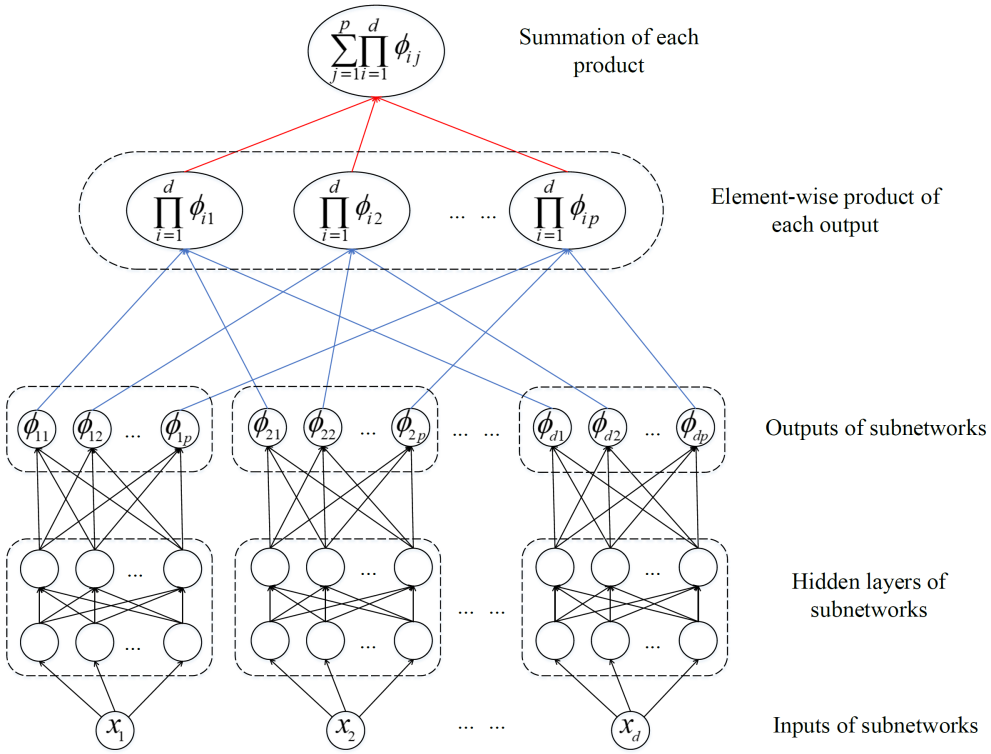

Fig. 2.2. The architecture of one-dimensional output TNN. Black arrows mean linear transformation (or affine transformation). Each ending node of blue arrows is obtained by taking the scalar multiplication of all starting nodes of blue arrows that end in this ending node. The final output of TNN is derived from the summation of all starting nodes of red arrows.

2.2. Approximation of TNN in Sobolev space

In order to show the validity of solving PDEs by TNN, we introduce the following approximation result for the functions in the space $H^m(\Omega_1 \times \cdots \times \Omega_d)$ under the sense of H^m-norm.

Theorem 2.1. *Assume that each Ω_i is a bounded open interval in $\mathbb{R}$ for $i = 1, \ldots, d$, $\Omega = \Omega_1 \times \cdots \times \Omega_d$, and the function $f(x) \in H^m(\Omega)$. Then for any tolerance $\varepsilon > 0$, there exist a positive integer p and the corresponding TNN defined by (2.3) such that the following approximation property holds:*

$$\|f(x) - \Psi(x;\theta)\|_{H^m(\Omega)} < \varepsilon. \tag{2.4}$$

Proof. Due to the isomorphism relation

$$H^m(\Omega) \cong H^m(\Omega_1) \otimes \cdots \otimes H^m(\Omega_d),$$

for any $\varepsilon > 0$ there exist a positive integer $p, h_{i,j}(x_i) \in H^m(\Omega_i), i = 1, \ldots, d$, $j = 1, \ldots, p$, and $h(x) \in H^m(\Omega)$, which is defined as follows:

$$h(x) = \sum_{j=1}^{p} h_{1,j}(x_1) \cdots h_{d,j}(x_d) = \sum_{j=1}^{p} \prod_{i=1}^{d} h_{i,j}(x_i),$$

such that the following estimate holds:

$$\|f(x) - h(x)\|_{H^m(\Omega)} < \frac{\varepsilon}{2}. \tag{2.5}$$

Denote $M_j = \max_i \|h_{i,j}(x_i)\|_{H^m(\Omega_i)}, j = 1, \ldots, p$, and set

$$M := \sum_{j=1}^{p} \left(\binom{d}{1} M_j^{d-1} + \binom{d}{2} M_j^{d-2} + \cdots + \binom{d}{d-1} M_j^1 + 1 \right).$$

From the density results [9, Chapter 5.3.3] and Ω_i is a bounded open interval in $\mathbb{R}$, there exists a $\bar{h}_{i,j}(x_i) \in C^\infty(\bar{\Omega}_i) \subset C(\bar{\Omega}_i)$ can approximate the one-dimensional function $h_{i,j}(x_i)$ with arbitrary accuracy under $H^m(\Omega)$-norm. It is shown in [8, 26] that one-hidden layer FNN can approximate any continuous function on a compact set as long as the activation function is not a polynomial. This conclusion can be naturally generalized from $\bar{\Omega}_i \to \mathbb{R}$ to $\bar{\Omega}_i \to \mathbb{R}^p$. Then for $\delta = \min\{1, \varepsilon/(2M)\}$, there exist FNN structures $\phi_i(x_i; \theta_i), i = 1, \ldots, d$, which are defined by (2.1), such that

$$\|h_{i,j}(x_i) - \phi_{i,j}(x_i; \theta_i)\|_{H^m(\Omega_i)} < \delta, \quad i = 1, \ldots, d, \quad j = 1, \ldots, p. \tag{2.6}$$

Denote $e_{i,j}(x_i) = \phi_{i,j}(x_i; \theta_i) - h_{i,j}(x_i)$, inequalities in (2.6) imply that $\|e_{i,j}(x_i)\|_{H^m(\Omega_i)} < \delta$.

Since the property of multidimensional integrations on the tensor product domain Ω, for any $g_i(x_i) \in H^m(\Omega_i), i = 1, \ldots, d$, the following inequality holds:

$$\left\| \prod_{i=1}^{d} g_i(x_i) \right\|_{H^m(\Omega)} \le \prod_{i=1}^{d} \|g_i(x_i)\|_{H^m(\Omega_i)}. \tag{2.7}$$

For the sake of clarity, we give a simple proof for (2.7) as follows:

$$\begin{aligned}
\left\| \prod_{i=1}^{d} g_i(x_i) \right\|_{H^m(\Omega)}^2 &= \sum_{|\alpha| \le m} \left\| D^\alpha \left(\prod_{i=1}^{d} g_i(x_i) \right) \right\|_{L^2(\Omega)}^2 = \sum_{\alpha_1 + \cdots + \alpha_d \le m} \left\| \prod_{i=1}^{d} \frac{\partial^{\alpha_i} g_i(x_i)}{\partial x_i^{\alpha_i}} \right\|_{L^2(\Omega_i)}^2 \\
&= \sum_{\alpha_1 + \cdots + \alpha_d \le m} \prod_{i=1}^{d} \left\| \frac{\partial^{\alpha_i} g_i(x_i)}{\partial x_i^{\alpha_i}} \right\|_{L^2(\Omega_i)}^2 \le \prod_{i=1}^{d} \left(\sum_{\alpha_i \le m} \left\| \frac{\partial^{\alpha_i} g_i(x_i)}{\partial x_i^{\alpha_i}} \right\|_{L^2(\Omega_i)}^2 \right) \\
&= \prod_{i=1}^{d} \|g_i(x_i)\|_{H^m(\Omega_i)}^2 .
\end{aligned}$$

Then from the property of binomial multiplication and inequality (2.7), we can build a TNN $\Psi(x; \theta)$ by (2.3) such that the following inequalities hold:

$$\begin{aligned}
&\|h(x) - \Psi(x; \theta)\|_{H^m(\Omega)} \\
&= \left\| \sum_{j=1}^{p} \prod_{i=1}^{d} h_{i,j}(x_i) - \sum_{j=1}^{p} \prod_{i=1}^{d} \big(h_{i,j}(x_i) + e_{i,j}(x_i) \big) \right\|_{H^m(\Omega)} \\
&\le \sum_{j=1}^{p} \left\| \prod_{i=1}^{d} h_{i,j}(x_i) - \prod_{i=1}^{d} \big(h_{i,j}(x_i) + e_{i,j}(x_i) \big) \right\|_{H^m(\Omega)} \\
&\le \sum_{j=1}^{p} \left(\binom{d}{1} M_j^{d-1} \delta^1 + \binom{d}{2} M_j^{d-2} \delta^2 + \cdots \binom{d}{d} M_j^0 \delta^d \right)
\end{aligned}$$

$$\leq \sum_{j=1}^{p} \left(\binom{d}{1} M_j^{d-1} + \binom{d}{2} M_j^{d-2} + \cdots + \binom{d}{d-1} M_j^1 + 1 \right) \delta$$
$$\leq M\delta < \frac{\varepsilon}{2}. \tag{2.8}$$

Therefore, from (2.5), (2.8) and triangle inequality, we have following estimates:

$$\|f(x) - \Psi(x;\theta)\|_{H^m(\Omega)} \leq \|f(x) - h(x)\|_{H^m(\Omega)} + \|h(x) - \Psi(x;\theta)\|_{H^m(\Omega)} < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

This is the desired result (2.4) and the proof is complete. □

Theorem 2.1 gives the approximation property of TNN, it shows that TNN can approximate any $H^m(\Omega)$ function under $H^m(\Omega)$-norm. It has to be pointed out that, since TNN has a tensor structure, each sub-network has only one-dimensional input. In the proof of Theorem 2.1, we only need the approximation property of FNNs with one-dimensional input. Compared with that of the FNNs with the d-dimensional input, the analysis of that with one-dimensional input is always easier. Here we cite a few instructive conclusions. In [16], it is shown that linear finite element basis can be represented by the FNN with one-dimensioanl input and the activation function being defined by the rectified linear unit (ReLU). In [27], it is proved that the monomial $x^n, n \in \mathbb{N}$ can be represented exactly by the FNN with the rectified power unit (ReQU) acting as the activation function.

2.3. Approximation in $H^{t,\ell}_{\mathbf{mix}}(\Omega)$ by TNN

Although there is no general result to give the relationship between the hyperparameter p and error bounds, there are still some estimations of traditional methods that can be used. For example, the sparse grid method and hyperbolic cross approximation method have become widely-used numerical tools for high-dimensional problems [34, 35]. These two methods also assume that the approximation function has a similar tensor-product form, while each one-dimensional function is defined on the linear space with fixed basis. The conclusions about the cardinal of subspaces for the sparse grid method and hyperbolic cross approximation method can be extended to the analysis of the hyperparameter p of TNN.

For clarity, we focus on the periodic setting with $I^d = I \times I \times \cdots \times I = [0, 2\pi]^d$ and the approximations property of TNN to the functions in the linear space which is defined with Fourier basis. Note that similar approximation results of TNN can be extended to the non-periodic functions.

First, for each variable $x_i \in [0, 2\pi]$, let us define the one-dimensional Fourier basis $\{\varphi_{k_i}(x_i) := e^{\mathrm{i}k_i x_i}/\sqrt{2\pi}, k_i \in \mathbb{Z}\}$ and classify functions via the decay of their Fourier coefficients. For example, the isotropic Sobolev spaces on I can be defined as follows [1]:

$$H^s(I) := \left\{ u(x_i) = \sum_{k_i \in \mathbb{Z}} c_{k_i} \varphi_{k_i}(x_i) : \|u\|_{H^s(I)} = \left(\sum_{k_i \in \mathbb{Z}} (1 + |k_i|)^{2s} \cdot |c_{k_i}|^2 \right)^{\frac{1}{2}} < \infty \right\}. \tag{2.9}$$

Further, denote multi-index $k = (k_1, \cdots, k_d) \in \mathbb{Z}^d$ and $x = (x_1, \cdots, x_d) \in I^d$. Then the d-dimensional Fourier basis can be built with the tensor product as

$$\varphi_k(x) \coloneqq \prod_{i=1}^{d} \varphi_{k_1}(x_i) = (2\pi)^{-\frac{d}{2}} e^{\mathrm{i}k \cdot x}. \tag{2.10}$$

We denote

$$\lambda_{\text{mix}}(k) \coloneqq \prod_{i=1}^{d}(1+|k_i|), \quad \lambda_{\text{iso}}(k) \coloneqq 1+\sum_{i=1}^{d}|k_i|. \tag{2.11}$$

Now, for $-\infty < t, \ell < \infty$, we define the space $H^{t,\ell}_{\text{mix}}(I^d)$ as follows (cf. [10, 21]):

$$H^{t,\ell}_{\text{mix}}(I^d) \coloneqq \left\{ u(x) = \sum_{k\in\mathbb{Z}^d} c_k\varphi_k(x) : \|u\|_{H^{t,\ell}_{\text{mix}}(I^d)} = \left(\sum_{k\in\mathbb{Z}^d} \lambda_{\text{mix}}(k)^{2t}\cdot\lambda_{\text{iso}}(k)^{2\ell}\cdot|c_k|^2 \right)^{\frac{1}{2}} < \infty \right\}. \tag{2.12}$$

Then the standard isotropic Sobolev spaces [1] and the Sobolev space of dominating mixed smoothness [33] can be written as

$$H^s(I^d) \coloneqq H^{0,s}_{\text{mix}}(I^d) = \left\{ u(x) = \sum_{k\in\mathbb{Z}^d} c_k\varphi_k(x) : \|u\|_{H^s_{\text{mix}}(I^d)} = \left(\lambda_{\text{iso}}(k)^{2s}\cdot|c_k|^2\right)^{\frac{1}{2}} < \infty \right\}, \tag{2.13}$$

and

$$\begin{aligned} H^t_{\text{mix}}(I^d) &\coloneqq H^{t,0}_{\text{mix}}(I^d) \\ &= \left\{ u(x) = \sum_{k\in\mathbb{Z}^d} c_k\varphi_k(x) : \|u\|_{H^{t,0}_{\text{mix}}(I^d)} = \left(\sum_{k\in\mathbb{Z}^d} \lambda_{\text{mix}}(k)^{2t}\cdot|c_k|^2 \right)^{\frac{1}{2}} < \infty \right\}, \end{aligned} \tag{2.14}$$

respectively. Note that the parameter ℓ governs the isotropic smoothness, whereas t governs the mixed smoothness. The space $H^{t,\ell}_{\text{mix}}(I^d)$ gives a quite flexible framework for the study of problems in Sobolev spaces. See [10, 11, 21] for more information on the space $H^{t,\ell}_{\text{mix}}(I^d)$.

Second, for $K \in \mathbb{N}$ and $T \in (-\infty, 1]$, define the following general sparse grid space to approximate functions in space $H^{t,\ell}_{\text{mix}}(I^d)$ according to the frequency k:

$$V_{K,T} \coloneqq \text{span}\{\varphi_k(x) : k \in \mathbb{Z}^d, \lambda_{\text{mix}}(k)\cdot\lambda_{\text{iso}}(k)^{-T} \le K^{1-T}\}. \tag{2.15}$$

The corresponding multi-index set of frequency k is

$$D_{K,T} \coloneqq \{k = (k_1, \cdots, k_d) : \lambda_{\text{mix}}(k)\cdot\lambda_{\text{iso}}(k)^{-T} \le K^{1-T}\}. \tag{2.16}$$

Obviously, the degree of freedom of space $|V_{K,T}|$ and the cardinal of the set $|D_{K,T}|$ are equivalent. By [22,39], the degree of freedom of space $V_{K,T}$ with respect to the parameter K and T is

$$|V_{T,K}| = \begin{cases} \mathcal{O}(K+1), & 0 < T < 1, \\ \mathcal{O}\big((K+1)\cdot\log(K+1)^{d-1}\big), & T = 0, \\ \mathcal{O}\left((K+1)^{\frac{T-1}{T/d-1}}\right), & T < 0, \\ \mathcal{O}\big((K+1)^d\big), & T = -\infty. \end{cases} \tag{2.17}$$

In this case of $0 < T < 1$, the degree of freedom of the space $V_{K,T}$ as well as the cardinal of the set $D_{K,T}$ are independent of dimension d. The following lemma gives the approximation property of the space $V_{K,T}$.

Lemma 2.1. *Let $f \in H^{t,\ell}_{mix}(I^d)$, $f_{K,T}$ be the best approximation in $V_{K,T}$ with respect to H^m-norm. Furthermore denote by p the actual number of degrees of freedom of $V_{K,T}$ as well as the cardinal of set $D_{K,T}$. Consider the case $T \in (0, (m-\ell)/t]$. Then, there holds*

$$\|f - f_{K,T}\|_{H^m(I^d)} \le C(d)\cdot p^{-(\ell-m+t)}\cdot\|u\|_{H^{t,\ell}_{mix}(I^d)}, \tag{2.18}$$

where $C(d) \le c\cdot d^2\cdot 0.97515^d$ and the constant c is independent of d.

Lemma 2.1 was introduced in [10], where more general cases are considered. Note that in Lemma 2.1, the best approximation $u_{K,T}$ has the following form:

$$f_{K,T} = \sum_{k\in D_{K,T}} c_k\varphi_k(x) = \sum_{k\in D_{K,T}} c_k \prod_{i=1}^{d} \varphi_{k_i}(x_i), \tag{2.19}$$

which is similar to the structure of TNN (2.3). Analogically, the hyperparameter p in TNN is equivalent to the cardinal of set $D_{K,T}$ and each one-dimensional Fourier basis $\varphi_{k_i}(x_i)$ is equivalent to $\phi_{i,j}(x_i)$ in (2.3). That is why we denote p in Lemma 2.1 as the cardinal of set $D_{K,T}$.

In order to obtain a comprehensive error estimate for TNN, the problem we left behind is whether an one-dimensional Fourier basis function can be approximated by an FNN with one-dimensional input. Fortunately, the Fourier basis function

$$\varphi_{k_i}(x_i) = \frac{1}{\sqrt{2\pi}} e^{-\mathrm{i}k_i x_i}$$

can be represented by the FNN with one-dimensional input, one hidden layer and activation function $\sigma(x) = \sin(x)$. The reason is based on the following property:

$$e^{-\mathrm{i}k_i x_i} = \cos(k_i x_i) - \mathrm{i}\sin(k_i x_i) = \sin\left(\frac{\pi}{2} - k_i x_i\right) - \mathrm{i}\sin(k_i x_i).$$

Thus, we can immediately obtain the following comprehensive error estimate for TNN.

Theorem 2.2. *Assume function $f(x) \in H^{t,\ell}_{mix}(I^d), t > 0$ and $m > \ell$. Then there exists a TNN $\Psi(x;\theta)$ defined by* (2.3) *such that the following approximation property holds:*

$$\|f(x) - \Psi(x;\theta)\|_{H^m(I^d)} \le C(d)\cdot p^{-(\ell-m+t)} \cdot \|u\|_{H^{t,\ell}_{mix}(I^d)}, \tag{2.20}$$

where $C(d) \le c\cdot d^2\cdot 0.97515^d$ and c is independent of d. And each subnetwork of TNN is an FNN which is built by using $\sin(x)$ *as the action function and one hidden layer with $2p$ neurons, see Fig.* 2.2.

The TNN-based machine learning method in this paper will adaptively select p rank-one functions by training process. From the approximation result in Theorem 2.2, when the target function belongs to $H^{t,\ell}_{\mathrm{mix}}(\Omega)$, there exists a TNN with $p \sim \mathcal{O}(\varepsilon^{-(m-\ell-t)})$ such that the accuracy is ε.

Note that our analysis is based on the special activation function $\sin(x)$ and space $H^{t,\ell}_{\mathrm{mix}}(\Omega)$. General approximation results in the $H^m(\Omega)$-norm for the functions in Barron space by the FNNs with d-dimensional input and general activation functions are discussed in [36].

3. Quadrature Scheme for TNN

In this section, we focus on the numerical integration of the polynomial composite function of TNN and its derivatives. Our main theorem shows that the application of TNN can bring a significant reduction of the computational complexity for the related numerical integration. For convenience, we first introduce the following sets of multiple indices:

$$\mathcal{B} := \left\{ \beta = (\beta_1, \cdots, \beta_d) \in \mathbb{N}_0^d \;\middle|\; |\beta| := \sum_{i=1}^{d} \beta_i \le m \right\},$$

$$\mathcal{A} := \left\{ \alpha = (\alpha_\beta)_{\beta\in\mathcal{B}} \in \mathbb{N}_0^{|\mathcal{B}|} \;\middle|\; |\alpha| := \sum_{\beta\in\mathcal{B}} \alpha_\beta \le k \right\},$$

where $\mathbb{N}_0$ denotes the set of all non-negative integers, m and k are two positive integers, $|\mathcal{B}|$ and $|\mathcal{A}|$ denote the cardinal numbers of $\mathcal{B}$ and $\mathcal{A}$, respectively.

For example, if $d = 2$ and $m = 1$, the set $\mathcal{B}$,

$$\mathcal{B} = \big\{(0,0),(1,0),(0,1)\big\}, \tag{3.1}$$

has 3 terms. Then $\mathcal{A}$ is a triple-index set. If $k = 2$, the set $\mathcal{A}$ can be described as follows:

$$\mathcal{A}=\big\{(0,0,0),(1,0,0),(0,1,0),(0,0,1),(2,0,0),(0,2,0),(0,0,2),(1,1,0),(1,0,1),(0,1,1)\big\}. \tag{3.2}$$

Each index in $\mathcal{A}$ corresponds to a member of $\mathcal{B}$. For example, we can simply take the order for the members in the set $\mathcal{B}$ as that of (3.1). Then the member $\alpha = (1,1,0) \in \mathcal{A}$ indicates that $\alpha_{(0,0)} = 1$, $\alpha_{(1,0)} = 1$ and $\alpha_{(0,1)} = 0$, the member $\alpha = (2,0,0) \in \mathcal{A}$ indicates $\alpha_{(0,0)} = 2, \alpha_{(1,0)} = 0$ and $\alpha_{(0,1)} = 0$.

In this paper, we focus on the high-dimensional cases where $m \ll d$ and $k \ll d$. Simple calculation leads to the following equations:

$$|\mathcal{B}| = \sum_{j=0}^{m} \binom{j+d-1}{j}, \quad |\mathcal{A}| = \sum_{j=0}^{k} \binom{j+|\mathcal{B}|-1}{j}.$$

By further estimation, we know that the scales of magnitudes of $|\mathcal{B}|$ and $|\mathcal{A}|$ are $\mathcal{O}((d+m)^m)$ and $\mathcal{O}(((d+m)^m + k)^k)$, respectively.

Here and after, the parameter θ in (2.3) will be omitted for brevity without confusion. The activation function of TNN needs to be smooth enough such that $\Psi(x)$ has partial derivatives up to order m. Here, we assume $F(x)$ includes the k-degree complete polynomial of d-dimensional TNN and its partial derivatives up to order m that can be expressed as follows:

$$F(x) = \sum_{\alpha\in\mathcal{A}} A_\alpha(x) \prod_{\beta\in\mathcal{B}} \left(\frac{\partial^{|\beta|}\Psi(x)}{\partial x_1^{\beta_1}\cdots\partial x_d^{\beta_d}} \right)^{\alpha_\beta}, \tag{3.3}$$

where the coefficient $A_\alpha(x)$ is given by the following expansion such that the rank of $A_\alpha(x)$ is not greater than q in the tensor product space $L^2(\Omega_1)\otimes\cdots\otimes L^2(\Omega_d)$:

$$A_\alpha(x) = \sum_{\ell=1}^{q} B_{1,\ell,\alpha}(x_1) B_{2,\ell,\alpha}(x_2)\cdots B_{d,\ell,\alpha}(x_d). \tag{3.4}$$

Here $B_{i,\ell,\alpha}(x_i)$ denotes the one-dimensional function in $L^2(\Omega_i)$ for $i = 1,\ldots,d$ and $\ell = 1,\ldots,q$. When using neural networks to solve PDEs, we always need to do the high-dimensional integration $\int_\Omega F(x)dx$. If $\Psi(x)$ is an FNN, $\int_\Omega F(x)dx$ can only be treated as a direct d-dimensional numerical integration, which requires an exponential scale of computational work according to the dimension d. In practical applications, it is well known that the high-dimensional FNN functions can only be integrated by the Monte-Carlo method with low accuracy. Different from FNN, we will show that the high-dimensional integration $\int_\Omega F(x)dx$ for the TNN can be implemented by the normal numerical quadrature with the polynomial scale of computational work with respect to the dimension d. This means that the TNN can cope with the curse of dimensionality in some sense. The key idea to reduce the computational complexity of the numerical integration $\int_\Omega F(x)dx$ is that we can decompose the TNN function $F(x)$ into a tensor product structure.

To implement the decomposition, for each $\alpha = (\alpha_1, \cdots, \alpha_{|\mathcal{B}|}) \in \mathcal{A}$, we give the following definition:

$$\mathcal{B}_\alpha := \big\{\beta = (\beta_1, \cdots, \beta_d) \in \mathcal{B} \;\big|\; \alpha_\beta \geq 1\big\}.$$

For example, when the sets $\mathcal{B}$ and $\mathcal{A}$ are defined by (3.1) and (3.2), respectively, the set $\mathcal{B}_\alpha$ corresponding to the member $\alpha = (1, 1, 0) \in \mathcal{A}$ in (3.2) can be described as follows:

$$\mathcal{B}_\alpha = \{(0, 0), (1, 0)\}.$$

By the definition of the index set $\mathcal{A}$, we can deduce that $|\mathcal{B}_\alpha| \leq k$ for any $\alpha \in \mathcal{A}$.

Since $\Psi(x)$ has the TNN structure (2.3), the compound can be further decomposed as

$$\begin{aligned}
&\prod_{\beta\in\mathcal{B}_\alpha}\left(\frac{\partial^{|\beta|}\Psi(x)}{\partial x_1^{\beta_1}\cdots\partial x_d^{\beta_d}}\right)^{\alpha_\beta}\\
&=\prod_{\beta\in\mathcal{B}_\alpha}\left(\frac{\partial^{|\beta|}\sum\limits_{j=1}^{p}\phi_{1,j}(x_1)\cdots\phi_{d,j}(x_d)}{\partial x_1^{\beta_1}\cdots\partial x_d^{\beta_d}}\right)^{\alpha_\beta}\\
&=\prod_{\beta\in\mathcal{B}_\alpha}\left(\sum_{j=1}^{p}\frac{\partial^{\beta_1}\phi_{1,j}(x_1)}{\partial x_1^{\beta_1}}\cdots\frac{\partial^{\beta_d}\phi_{d,j}(x_d)}{\partial x_d^{\beta_d}}\right)^{\alpha_\beta}\\
&=\prod_{\beta\in\mathcal{B}_\alpha}\sum_{1\leq j_1,\cdots,j_{\alpha_\beta}\leq p}\left(\frac{\partial^{\beta_1}\phi_{1,j_1}(x_1)}{\partial x_1^{\beta_1}}\cdots\frac{\partial^{\beta_1}\phi_{1,j_{\alpha_\beta}}(x_1)}{\partial x_1^{\beta_1}}\right)\cdots\left(\frac{\partial^{\beta_d}\phi_{d,j_1}(x_d)}{\partial x_d^{\beta_d}}\cdots\frac{\partial^{\beta_d}\phi_{d,j_{\alpha_\beta}}(x_d)}{\partial x_d^{\beta_d}}\right)\\
&=\prod_{\beta\in\mathcal{B}_\alpha}\sum_{1\leq j_1,\cdots,j_{\alpha_\beta}\leq p}\left(\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_1}\phi_{1,j_\ell}(x_1)}{\partial x_1^{\beta_1}}\right)\cdots\left(\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_d}\phi_{d,j_\ell}(x_d)}{\partial x_d^{\beta_d}}\right)\\
&=\sum_{\substack{\beta\in\mathcal{B}_\alpha,\ell=1,\ldots,\alpha_\beta,\\ 1\leq j_{\beta,\ell}\leq p}}\left(\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_1}\phi_{1,j_{\beta,\ell}}(x_1)}{\partial x_1^{\beta_1}}\right)\cdots\left(\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_d}\phi_{d,j_{\beta,\ell}}(x_d)}{\partial x_d^{\beta_d}}\right).
\end{aligned}\tag{3.5}$$

With the help of expansion (3.5), we can give the following expansion for $F(x)$:

$$\begin{aligned}
F(x)&=\sum_{\alpha\in\mathcal{A}}\left(\sum_{\ell=1}^{q}B_{1,\ell,\alpha}(x_1)\cdots B_{d,\ell,\alpha}(x_d)\right)\\
&\quad\times\sum_{\substack{\beta\in\mathcal{B}_\alpha,\ell=1,\ldots,\alpha_\beta,\\ 1\leq j_{\beta,\ell}\leq p}}\left(\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_1}\phi_{1,j_{\beta,\ell}}(x_1)}{\partial x_1^{\beta_1}}\right)\cdots\left(\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_d}\phi_{d,j_{\beta,\ell}}(x_d)}{\partial x_d^{\beta_d}}\right)\\
&=\sum_{\alpha\in\mathcal{A}}\sum_{\ell=1}^{q}\sum_{\substack{\beta\in\mathcal{B}_\alpha,\ell=1,\ldots,\alpha_\beta,\\ 1\leq j_{\beta,\ell}\leq p}}\left(B_{1,\ell,\alpha}(x_1)\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_1}\phi_{1,j_{\beta,\ell}}(x_1)}{\partial x_1^{\beta_1}}\right)\\
&\qquad\cdots\left(B_{d,\ell,\alpha}(x_d)\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_d}\phi_{d,j_{\beta,\ell}}(x_d)}{\partial x_d^{\beta_d}}\right).
\end{aligned}\tag{3.6}$$

Based on the decomposition (3.6), we have the following splitting scheme for the integration $\int_\Omega F(x)dx$:

$$\int_{\Omega} F(x)dx = \sum_{\alpha\in\mathcal{A}}\sum_{\ell=1}^{q}\sum_{\substack{\beta\in\mathcal{B}_\alpha,\ell=1,\ldots,\alpha_\beta,\\ 1\le j_{\beta,\ell}\le p}}\int_{\Omega_1}\left(B_{1,\ell,\alpha}(x_1)\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_1}\phi_{1,j_{\beta,\ell},\ell}(x_1)}{\partial x_1^{\beta_1}}\right)dx_1$$
$$\cdots\int_{\Omega_d}\left(B_{d,\ell,\alpha}(x_d)\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_d}\phi_{d,j_{\beta,\ell},\ell}(x_d)}{\partial x_d^{\beta_d}}\right)dx_n. \quad (3.7)$$

Now, we introduce the detailed numerical integration method for the TNN function $F(x)$. Without loss of generality, for $i = 1,\ldots,d$, we choose N_i Gauss points $\{x_i^{(n_i)}\}_{n_i=1}^{N_i}$ and the corresponding weights $\{w_i^{(n_i)}\}_{n_i=1}^{N_i}$ for the i-th dimensional domain Ω_i, and denote $N = \max\{N_1,\cdots,N_d\}$ and $\underline{N} = \min\{N_1,\cdots,N_d\}$. Introducing the index $n = (n_1,\cdots,n_d) \in \mathcal{N} := \{1,\ldots,N_1\}\times\cdots\times\{1,\ldots,N_d\}$, then the Gauss points and their corresponding weights for the integration (3.7) can be expressed as follows:

$$\begin{aligned}\{x^{(n)}\}_{n\in\mathcal{N}} &= \left\{\{x_1^{(n_1)}\}_{n_1=1}^{N_1},\{x_2^{(n_2)}\}_{n_2=1}^{N_2},\cdots,\{x_d^{(n_d)}\}_{n_d=1}^{N_d}\right\},\\ \{w^{(n)}\}_{n\in\mathcal{N}} &= \left\{\{w_1^{(n_1)}\}_{n_1=1}^{N_1}\times\{w_2^{(n_2)}\}_{n_2=1}^{N_2}\times\cdots,\times\{w_d^{(n_d)}\}_{n_d=1}^{N_d}\right\}.\end{aligned} \quad (3.8)$$

Then from (3.3) and (3.4), the numerical integration $\int_\Omega F(x)dx$ can be computed as follows:

$$\int_{\Omega} F(x)dx \approx \sum_{n\in\mathcal{N}} w^{(n)}\sum_{\alpha\in\mathcal{A}}\left(\sum_{\ell=1}^{q}B_{1,\ell,\alpha}(x_1^{(n_1)})\cdots B_{d,\ell,\alpha}(x_d^{(n_d)})\right)\prod_{\beta\in\mathcal{B}_\alpha}\left(\frac{\partial^{|\beta|}\Psi(x^{(n)})}{\partial x_1^{\beta_1}\cdots\partial x_d^{\beta_d}}\right)^{\alpha_\beta}. \quad (3.9)$$

Fortunately, with the help of expansions (3.5) and (3.7), we can give the following splitting numerical integration scheme for $\int_\Omega F(x)dx$:

$$\int_{\Omega} F(x)dx \approx \sum_{\alpha\in\mathcal{A}}\sum_{\ell=1}^{q}\sum_{\substack{\beta\in\mathcal{B}_\alpha,\ell=1,\ldots,\alpha_\beta,\\ 1\le j_{\beta,\ell}\le p}}\left(\sum_{n_1=1}^{N_1}w_1^{(n_1)}B_{1,\ell,\alpha}(x_1^{(n_1)})\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_1}\phi_{1,j_{\beta,\ell},\ell}(x_1^{(n_1)})}{\partial x_1^{\beta_1}}\right)$$
$$\cdots\left(\sum_{n_d=1}^{N_d}w_d^{(n_d)}B_{d,\ell,\alpha}(x_d^{(n_d)})\prod_{\beta\in\mathcal{B}_\alpha}\prod_{\ell=1}^{\alpha_\beta}\frac{\partial^{\beta_d}\phi_{d,j_{\beta,\ell},\ell}(x_d^{(n_d)})}{\partial x_d^{\beta_d}}\right). \quad (3.10)$$

The quadrature scheme (3.10) decomposes the high-dimensional integration $\int_\Omega F(x)dx$ into a series of one-dimensional integration, which is the main contribution of this paper. Due to the simplicity of the one-dimensional integration, the scheme (3.10) can reduce the computational work of the high-dimensional integration for the d-dimensional function $F(x)$ to the polynomial scale of dimension d. Theorem 3.1 gives the corresponding result.

Theorem 3.1. *Assume that the function $F(x)$ is defined as* (3.3)*, where the coefficient $A_\alpha(x)$ has the expansion* (3.4)*. Employ Gauss quadrature points and corresponding weights* (3.8) *to $F(x)$ on the d-dimensional tensor product domain Ω. If the function $\Psi(x)$ involved in the function $F(x)$ has TNN form* (2.3)*, the efficient quadrature scheme* (3.10) *is equivalent to* (3.9) *and has $2\underline{N}$-th order accuracy. Let T_1 denote the computational complexity for the one-dimensional function evaluation operations. The computational complexity can be bounded by $\mathcal{O}(dqT_1k^2p^k((d+m)^m+k)^kN)$, which is the polynomial scale of the dimension d.*

Proof. First, we point out that the number of $j_{\beta,\ell}$ in the last summation of (3.5) is no more than k. This result can be easily proved by the following inequality:

$$\sum_{\beta\in\mathcal{B}_\alpha}\alpha_\beta = |\alpha| \le k.$$

Then, by direct calculation, the computational complexity of (3.10) can be bounded by $\mathcal{O}(dqT_1\ k^2p^k((d+m)^m+k)^kN)$. Since the one-dimensional integration with N_i Gauss points has $2N_i$-th order accuracy and the equivalence of (3.10) and (3.9), both quadrature schemes (3.10) and (3.9) have the $2\underline{N}$-th order accuracy. The proof is complete. □

Remark 3.1. Theorem 3.1 considers using one-dimensional Gauss points to compute d-dimensional integrations. Other types of one-dimensional quadrature schemes can also be employed to do the d-dimensional integration and have similar results. In numerical examples, we decompose each Ω_i into subintervals with mesh size h and choose N_i one-dimensional Gauss points in each subinterval. Then the deduced d-dimensional quadrature scheme has accuracy $\mathcal{O}(h^{2\underline{N}}/(2\underline{N})!)$, where the included constant depends on the smoothness of $F(x)$.

If $\Psi(x)$ does not have the tensor form, for example, $\Psi(x)$ is a d-dimensional FNN and use the same quadrature scheme (3.10), the computational complexity is $\mathcal{O}((dqT_1+kT_d)((d+m)^m+k)^kN^d)$, where T_d denotes the complexity of the d-dimensional function evaluation operations.

4. Solving High-Dimensional Eigenvalue Problem by TNN

This section is devoted to discussing the applications of TNNs to the numerical solution of the high-dimensional second order elliptic eigenvalue problems. For simplicity, we are concerned with the following model problem:

$$\begin{cases} -\Delta u + vu = \lambda u & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{4.1}$$

where $\Omega = \Omega_1 \times \cdots \times \Omega_d$, each $\Omega_i = (a_i, b_i), i = 1, \ldots, d$ is a bounded interval in $\mathbb{R}$, $v \in L^2(\Omega)$ is a potential function. We assume that the rank of v is finite in the tensor product space $L^2(\Omega_1) \otimes \cdots \otimes L^2(\Omega_d)$. The potential function v often occurs in quantum mechanics problems. In this paper, we consider the following cases:

zero function

$$v(x) = 0 \quad \text{in } \Omega, \tag{4.2}$$

harmonic oscillator

$$v(x) = \sum_{i=1}^{d} x_i^2 \quad \text{in } \Omega, \tag{4.3}$$

coupled oscillator

$$v(x) = \sum_{i=1}^{d} x_i^2 - \sum_{i=1}^{d-1} x_i x_{i+1}, \tag{4.4}$$

and the Coulomb potential for Schrödinger equation.

In quantum mechanics, the eigenvalue problem (4.1) with the potential function (4.2) is the Schrödinger equation with infinite potential well. The eigenvalue problem with the potential (4.3) comes from the truncation of the Schrödinger equation with the harmonic oscillator potential which is defined in the whole space. The more complicated eigenvalue problem with the potential (4.4) describes the system of chains of d coupled harmonic oscillators which is described in detail in [3].

There is a well-known variational principle or minimum theorem of the eigenvalue problem (4.1) for the smallest eigenpair (λ, u)

$$\lambda = \min_{w\in H_0^1(\Omega)} \mathcal{R}(w) = \min_{w\in H_0^1(\Omega)} \frac{\int_\Omega |\nabla w|^2 dx + \int_\Omega v w^2 dx}{\int_\Omega w^2 dx}, \tag{4.5}$$

where $\mathcal{R}(w)$ denotes the Rayleigh quotient for the function $w \in H_0^1(\Omega)$.

In order to solve the eigenvalue problem (4.1), we build a TNN structure $\Psi(x;\theta)$ which is defined by (2.3), and denote the set of all possible values of θ as Θ. In order to avoid the penalty on boundary conditions, we simply use the method in [12] to treat the Dirichlet boundary condition. This method is firstly proposed in [24, 25]. For $i = 1, \ldots, d$, the i-th subnetwork $\phi_i(x_i;\theta_i)$ is defined as follows:

$$\begin{aligned}\phi_i(x_i;\theta_i) &:= (x_i - a_i)(b_i - x_i)\widehat{\phi}_i(x_i;\theta_i)\\ &= \big((x_i - a_i)(b_i - x_i)\widehat{\phi}_{i,1}(x_i;\theta_i), \cdots, (x_i - a_i)(b_i - x_i)\widehat{\phi}_{i,p}(x_i;\theta_i)\big)^T,\end{aligned}$$

where $\widehat{\phi}_i(x_i;\theta_i)$ is an FNN from $\mathbb{R}$ to $\mathbb{R}^p$ with sufficient smooth activation functions, such that $\Psi(x;\theta) \in H_0^1(\Omega)$.

The trial function set V is modeled by the TNN structure $\Psi(x;\theta)$ where parameters θ take all the possible values and it is obvious that $V \subset H_0^1(\Omega)$. The solution and the parameters $(\lambda^*, \Psi(x;\theta^*))$ of the following optimization problem are approximations to the smallest eigenpair:

$$\lambda^* = \min_{\Psi(x;\theta)\in V} \mathcal{R}\big(\Psi(x;\theta)\big) = \min_{\theta\in\Theta} \frac{\int_\Omega |\nabla\Psi(x;\theta)|^2 dx + \int_\Omega v(x)\Psi^2(x;\theta)dx}{\int_\Omega \Psi^2(x;\theta)dx} = \mathcal{R}\big(\Psi(x;\theta^*)\big). \tag{4.6}$$

Note that all integrands of the numerator and the denominator of (4.6) have the form (3.3). With the help of Theorem 3.1, we can implement these numerical integrations by the scheme (3.10) with the computational work being bounded by the polynomial scale of dimension d. We choose Gauss points and their corresponding weights which are defined by (3.8) to compute these integrations, and define the loss function as follows:

$$L(\theta) := \frac{\sum_{n\in\mathcal{N}} w^{(n)} |\nabla\Psi(x^{(n)};\theta)|^2 + \sum_{n\in\mathcal{N}} w^{(n)} v(x^{(n)})\Psi^2(x^{(n)};\theta)}{\sum_{n\in\mathcal{N}} w^{(n)}\Psi^2(x^{(n)};\theta)}. \tag{4.7}$$

In this paper, the gradient descent (GD) method is adopted to minimize the loss function $L(\theta)$. The GD scheme can be described as follows:

$$\theta^{(k+1)} = \theta^{(k)} - \eta\nabla L(\theta^{(k)}), \tag{4.8}$$

where $\theta^{(k)}$ denotes the parameters after the k-th GD step, η is the learning rate (step size).

Different from the general FNN-based machine learning method, we use the fixed quadrature points $\{x^{(n)}\}_{n\in\mathcal{N}}$ to do the numerical integration in this paper. Using the fixed quadrature points for FNN, the computational work for the numerical integration will depend exponentially on the dimension d. In order to avoid the "curse of dimensionality", in the numerical implementation for solving high-dimensional PDEs by FNN-based method, the stochastic gradient descent (SGD) method [20] with Monte-Carlo integration are always used [7]. The application of random sampling quadrature points always leads to low accuracy and instability convergence for the FNN method.

Fortunately, based on TNN structure in the loss function (4.7), Theorem 3.1 shows that the numerical integration does not encounter "curse of dimensionality" since the computational work can be bounded by the polynomial scale of dimension d. This is the reason we can use GD method to solve the optimization problem (4.6) instead of SGD in this paper. That is to say, using all quadrature points to implement the integration and the GD step (4.8) in TNN-based machine learning are reasonable. With the help of the high accuracy of the tensor product with Gauss points and Theorem 2.1, the high accuracy of the TNN-based method can be guaranteed.

Although in this paper, we simply choose a fixed rank p in our numerical examples, it is worth mentioning that, by adding columns to the weight matrices of the output layer in each subnetwork, we can transfer weights from the old TNN to the new one to improve the rank p. We can stop this process when the accuracy improvement is small enough. Choosing the rank p by the computable posterior error estimation and the corresponding transfer learning framework will be presented in our future work.

Remark 4.1. In this section, we are mainly talking about solving high-dimensional eigenvalue problems by TNN. It is worth mentioning that the TNN structure can be applied naturally to solve PDEs with different types of loss functions. We will do a preliminary test in our numerical examples.

5. Numerical Examples

In this section, we provide several examples to validate the efficiency and accuracy of the TNN-based machine learning method proposed in this paper. The first two examples are used to demonstrate the high accuracy of the TNN method for high-dimensional problems. We will explore the effect of the vital hyperparameter p on the accuracy in the third example where the ground state energy may not be exactly represented by a finite-rank CP decomposition. The fourth example for the ground state of a helium atom which comes from the real physical problem is used to give an illuminating way to deal with the problem that does not satisfy the assumption (3.4). Note that in the fourth example, the potential cannot be exactly expressed as a CP decomposition of finite rank, this makes the loss function no longer satisfy the assumption (3.4). In the last example, we solve a boundary value problem with Neumann boundary condition to show the efficiency of TNN for solving high-dimensional PDEs.

In order to show the convergence behavior and accuracy of eigenfunction approximations by TNN, we define the $L^2(\Omega)$ projection operator $\mathcal{P} : H_0^1(\Omega) \to \mathrm{span}\{\Psi(x;\theta^*)\}$ as follows:

$$\langle \mathcal{P}u, v\rangle_{L^2} = \langle u, v\rangle_{L^2} \coloneqq \int_\Omega uv dx, \quad \forall\, v \in \mathrm{span}\{\Psi(x;\theta^*)\}, \quad u \in H_0^1(\Omega).$$

And we define the $H^1(\Omega)$ projection operator $\mathcal{Q} : H_0^1(\Omega) \to \mathrm{span}\{\Psi(x;\theta^*)\}$ as follows:

$$\langle \mathcal{Q}u, v\rangle_{H^1} = \langle u, v\rangle_{H^1} \coloneqq \int_\Omega \nabla u \cdot \nabla v dx, \quad \forall\, v \in \mathrm{span}\{\Psi(x;\theta^*)\}, \quad u \in H_0^1(\Omega).$$

Then we define the following errors for the approximated eigenvalue λ^* and eigenfunction $\Psi(x;\theta^*)$:

$$e_\lambda \coloneqq \frac{|\lambda^* - \lambda|}{|\lambda|}, \quad e_{L^2} \coloneqq \frac{\|u - \mathcal{P}u\|_{L^2(\Omega)}}{\|u\|_{L^2(\Omega)}}, \quad e_{H^1} \coloneqq \frac{|u - \mathcal{Q}u|_{H^1(\Omega)}}{|u|_{H^1(\Omega)}}$$

in all eigenvalue examples. As for Neumann boundary value problem, we define the following errors for the approximated solution $\Psi(x;\theta^*)$:

$$\widehat{e}_{L^2} \coloneqq \frac{\|u-\Psi(x;\theta^*)\|_{L^2(\Omega)}}{\|f\|_{L^2(\Omega)}}, \quad \widehat{e}_{H^1} \coloneqq \frac{|u-\Psi(x;\theta^*)|_{H^1(\Omega)}}{|f|_{H^1(\Omega)}}.$$

Here $\|\cdot\|_{L^2}$ and $|\cdot|_{H^1}$ denote $L^2(\Omega)$-norm and $H^1(\Omega)$-seminorm, respectively. These relative errors are often used to test numerical methods for eigenvalue problems and PDEs. We use the quadrature scheme (3.10) to compute e_{L^2} and e_{H^1} with the same Gauss points and weights as computing the loss functions if the rank of the exact solution $u(x)$ is finite in the tensor product space $L^2(\Omega_1)\otimes\cdots\otimes L^2(\Omega_d)$, otherwise we only report e_λ. With the help of Theorem 3.1 and Gauss quadrature points, the high efficiency and accuracy for computing e_{L^2} and e_{H^1} can be guaranteed.

In implementation, we train the networks by Adam optimizer [20] and use automatic differentiation for derivatives in PyTorch. In this chapter, all examples is using sine function $\sin(x)$ as the activation function.

5.1. Laplace eigenvalue problem

In the first example, the potential function is defined as (4.2) with the computational domain $\Omega=[0,1]^d$. Then the exact smallest eigenvalue and eigenfunction are

$$\lambda = d\pi^2, \quad u(x) = \prod_{i=1}^{d}\sin(\pi x_i).$$

First, we test high-dimensional cases with $d=5,10,20$. Quadrature scheme for TNN is obtained by decomposing each $\Omega_i, i=1,\ldots,d$, into 10 equal subintervals and choosing 16 Gauss points on each subinterval. The Adam optimizer is employed with a learning rate of 0.003 to train a TNN with $p=10$. Each subnetwork of TNN is an FNN with two hidden layers and each hidden layer has 50 hidden neurons, see Fig. 2.2. Fig. 5.1 shows the relative errors e_λ, e_{L^2} and e_{H^1} versus the number of epochs. The final relative errors after 100000 epochs are reported in Table 5.1 for different dimensional cases. We can find that the TNN method has almost the same convergence behaviors for different dimensions.

Then we test ultra-high-dimensional cases with d=128, 256, 512. Although the usual problem does not have such a high dimension, it is important to point out that since the TNN structure (2.3) includes the compound of d terms, in ultra-high-dimensional cases, numerical instability may occur. To improve the numerical stability, we do a suitable scale for each dimension of TNN at the initialization step. In implementation, we decompose each Ω_i, $i=1,\ldots,d$, into 10 equal subintervals and choose 16 Gauss points on each subinterval. The Adam optimizer is employed with a learning rate 0.003 to train a smaller-scale TNN with $p=10$. Each subnetwork in of the TNN is an FNN with two hidden layers and each hidden layer has 20 hidden neurons. We use the Adam optimizer in the first 100000 steps and then the L-BFGS in the subsequent 10000 steps to produce the final results. The final results are shown in Fig. 5.2 and Table 5.2. In ultra-high-dimensional cases, the TNN method still has almost the same convergence behaviors for different dimensions, and the final results are not much worse than that in high-dimensional cases. All relative errors are on a convincing order of magnitude.

Table 5.1: Errors of Laplace eigenvalue problem for $d = 5, 10, 20$.

d	e_λ	e_{L^2}	e_{H^1}
5	4.838e-09	1.977e-05	7.231e-05
10	7.916e-09	8.941e-05	1.261e-04
20	6.354e-09	6.872e-05	1.052e-04

Table 5.2: Errors of Laplace eigenvalue problem for $d = 128, 256, 512$.

d	e_λ	e_{L^2}	e_{H^1}
128	5.462e-08	4.676e-04	5.223e-04
256	1.980e-08	3.962e-04	4.205e-04
512	1.560e-08	4.956e-04	5.111e-04

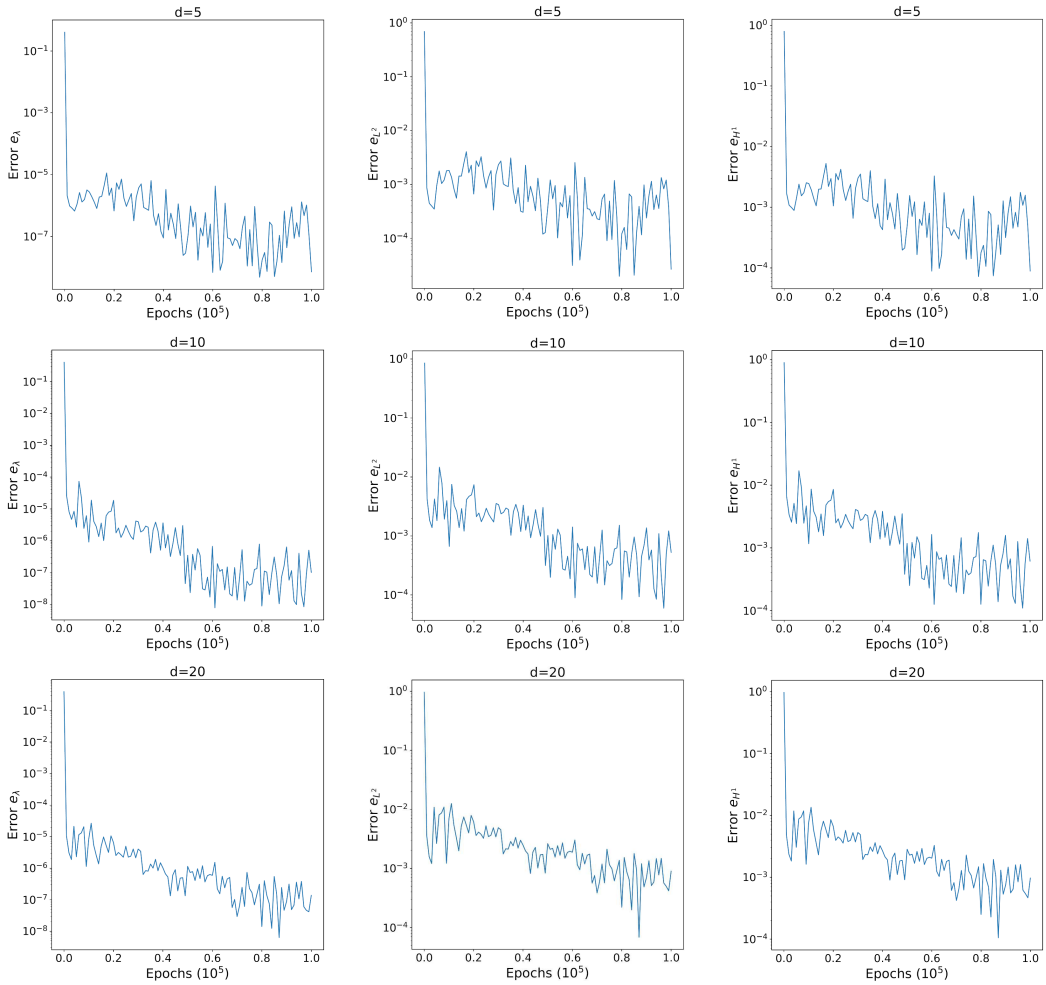

Fig. 5.1. Relative errors during the training process for Laplace eigenvalue problem, $d = 5, 10$, and 20. The left column shows the relative errors of eigenvalue approximations, the middle column shows the relative $L^2(\Omega)$ errors and the right column shows the relative $H^1(\Omega)$ errors of eigenfunction approximations.

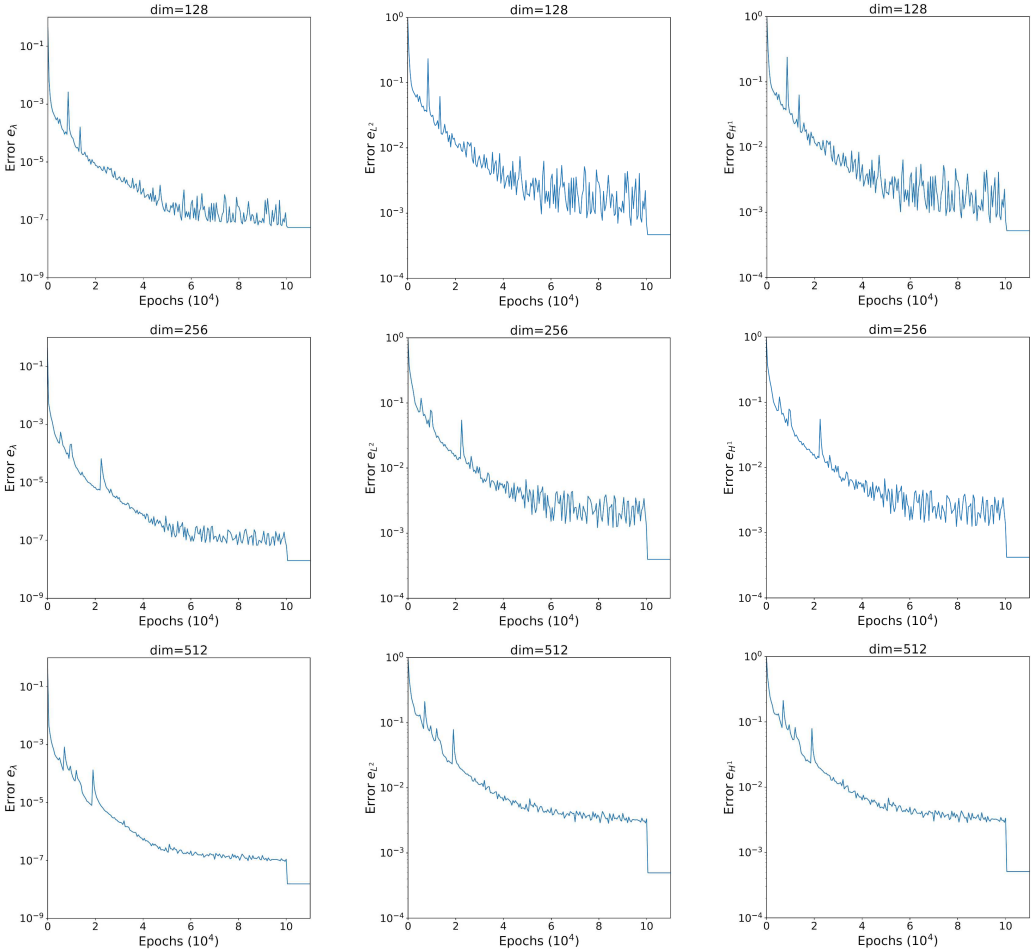

Fig. 5.2. Relative errors during the training process for Laplace eigenvalue problem, $d = 128, 256$ and 512. The left column shows the relative errors of eigenvalue approximations, the middle column shows the relative $L^2(\Omega)$ errors and the right column shows the relative $H^1(\Omega)$ errors of eigenfunction approximations.

5.2. Eigenvalue problem with harmonic oscillator

In the second example, the potential function is defined as (4.3). Then the exact smallest eigenvalue and eigenfunction are

$$\lambda = d, \quad u(x) = \prod_{i=1}^{d} e^{-\frac{x_i^2}{2}}.$$

As the first example in Section 5.1, high-dimensional cases with $d = 5, 10, 20$ and ultra-high-dimensional cases with $d = 128, 256, 512$ are tested, respectively. We truncate the computational domain from $\mathbb{R}^d$ to $[-5, 5]^d$, use 100 equal subintervals and 16 Gauss points quadrature scheme for all cases. The Adam optimizer is employed to train TNN of the same size as the first example but with a learning rate of 0.01 and 0.003 for high-dimensional cases and ultra-high-dimensional cases, respectively. For high-dimensional cases, the Adam optimizer is used 100000

steps. For ultra-high-dimensional cases, we use the Adam optimizer in the first 50000 steps and then the L-BFGS in the subsequent 10000 steps to produce the final results. Numerical results for $d = 5, 10, 20$ are shown in Fig. 5.3 and Table 5.3 and that for $d = 128, 256, 512$ are shown in Fig. 5.4 and Table 5.4. Since we truncate the computational domain, it is reasonable that the final relative errors are a little worse than the examples of the Laplace eigenvalue problem. There should exist some room for improving the accuracy.

Table 5.3: Errors of the harmonic oscillator problem for $d = 5, 10, 20$.

d	e_λ	e_{L^2}	e_{H^1}
5	4.241e-07	3.626e-04	8.431e-04
10	2.446e-07	2.709e-04	6.889e-04
20	7.225e-07	1.361e-03	1.555e-03

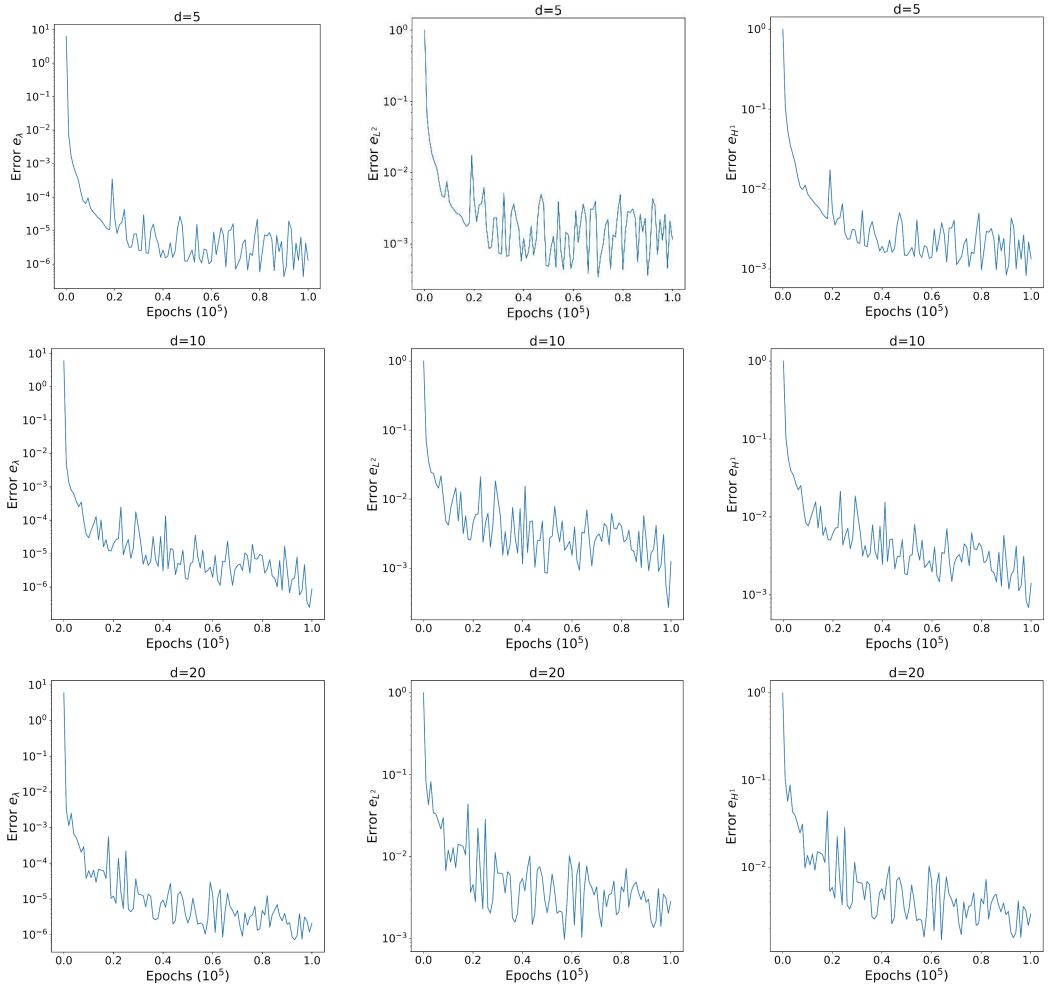

Fig. 5.3. Relative errors during the training process for the harmonic oscillator problem, $d = 5, 10, 20$. The left column shows the relative errors of eigenvalue, the middle column shows the relative L^2 errors and the right column shows the relative H^1 errors of eigenfunction approximations.

Table 5.4: Errors of the harmonic oscillator problem for $d = 128, 256, 512$.

d	e_λ	e_{L^2}	e_{H^1}
128	6.188e-07	2.979e-04	7.884e-04
256	1.080e-06	4.195e-04	1.181e-03
512	1.570e-06	5.570e-04	1.532e-03

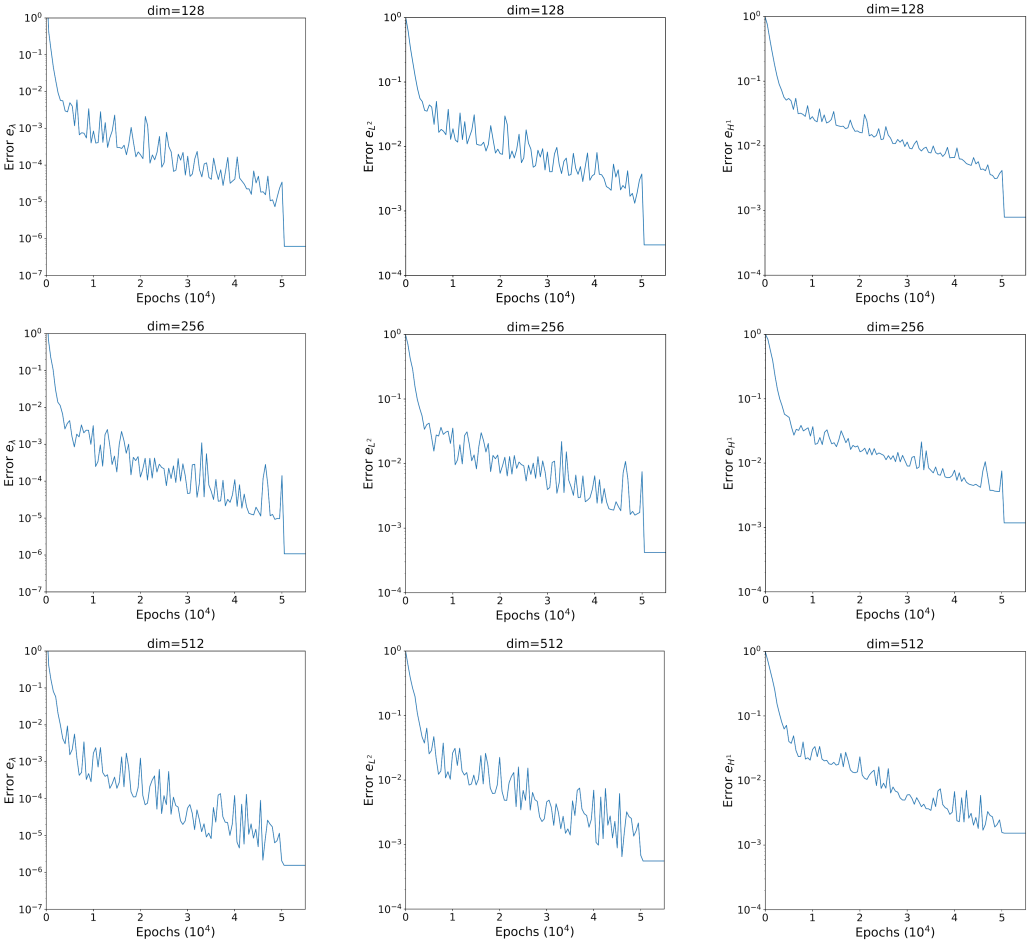

Fig. 5.4. Relative errors during the training process for the harmonic oscillator problem, $d = 128$, $256, 512$. The left column shows the relative errors of eigenvalue approximations, the middle column shows the relative L^2 errors and the right column shows the relative H^1 errors of eigenfunction approximations.

5.3. Eigenvalue problem with coupled harmonic oscillator

In the third example, the potential function is defined as (4.4). Similar to the derivation in [3], the exact smallest eigenvalue is

$$\lambda_0 = \sum_{i=1}^{d} \sqrt{1 - \cos\left(\frac{i\pi}{d+1}\right)},$$

and the exact eigenfunction has Gaussian form. In this example, we test the case of $d = 4$. Then the exact eigenfunction is

$$\begin{aligned}
&u(x_1, x_2, x_3, x_4)\\
&= \exp\Big[-\frac{1}{2}\big(\omega_1 a^2 + \omega_2 b^2 + \omega_3 a^2 + \omega_4 b^2\big)\big(x_1^2 + x_3^2\big)\\
&\quad -\frac{1}{2}\big(\omega_1 b^2 + \omega_2 a^2 + \omega_3 b^2 + \omega_4 a^2\big)\big(x_2^2 + x_4^2\big)\\
&\quad - ab(-\omega_1 - \omega_2 + \omega_3 + \omega_4)(x_1 x_2 + x_3 x_4)\\
&\quad - ab(\omega_1 - \omega_2 + \omega_3 - \omega_4)(x_1 x_4 + x_2 x_3)\\
&\quad - \big(-\omega_1 a^2 + \omega_2 b^2 + \omega_3 a^2 - \omega_4 b^2\big) x_1 x_3\\
&\quad - \big(-\omega_1 b^2 + \omega_2 a^2 + \omega_3 b^2 - \omega_4 a^2\big) x_2 x_4 \Big],
\end{aligned}$$

where

$$\begin{aligned}
&a = \frac{\sqrt{5-\sqrt{5}}}{2\sqrt{5}}, \quad b = \frac{\sqrt{5+\sqrt{5}}}{2\sqrt{5}},\\
&\omega_1 = \frac{\sqrt{5+\sqrt{5}}}{2}, \quad \omega_2 = \frac{\sqrt{3+\sqrt{5}}}{2},\\
&\omega_3 = \frac{\sqrt{5-\sqrt{5}}}{2}, \quad \omega_4 = \frac{\sqrt{3-\sqrt{5}}}{2}.
\end{aligned}$$

To demonstrate the efficiency of the proposed method, we take hyperparameter p from 1 to 30. Each dimension of the TNN is an FNN with two hidden layers and each hidden layer has 50 hidden neurons. We train the TNN to investigate the dependence of the convergence behavior on the hyperparameter p. All the cases are trained by the Adam optimizer with the same learning rate of 0.001 and epochs of 500000. The computational domain is truncated to $[-5, 5]^d$. And we decompose the interval $[-5, 5]$ in each dimensional into 100 equal subintervals and choose 16 Gauss points on each subinterval. The relative errors e_λ during the training process for different p are shown in Figs. 5.5 and 5.6 demonstrates how the final error changes as p increases. From Figs. 5.5 and 5.6, we can find that the proposed method converges at an impressive accuracy.

5.4. Ground state of helium atom

In the fourth example, we consider the Schrödinger equation of the helium atom whose potential cannot be exactly expressed as a CP decomposition of finite rank. The wave function of the helium atom with the fixed nucleus in Euclidean coordinates $\Psi(x_1, y_1, z_1, x_2, y_2, z_2)$ satisfies the following eigenvalue problem:

$$-\frac{1}{2}\Delta\Psi - \frac{2\Psi}{r_1} - \frac{2\Psi}{r_2} + \frac{\Psi}{r_{12}} = E\Psi, \tag{5.1}$$

where

$$\begin{aligned}
&r_1^2 = x_1^2 + y_1^2 + z_1^2, \quad r_2^2 = x_2^2 + y_2^2 + z_2^2,\\
&r_{12}^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2.
\end{aligned}$$

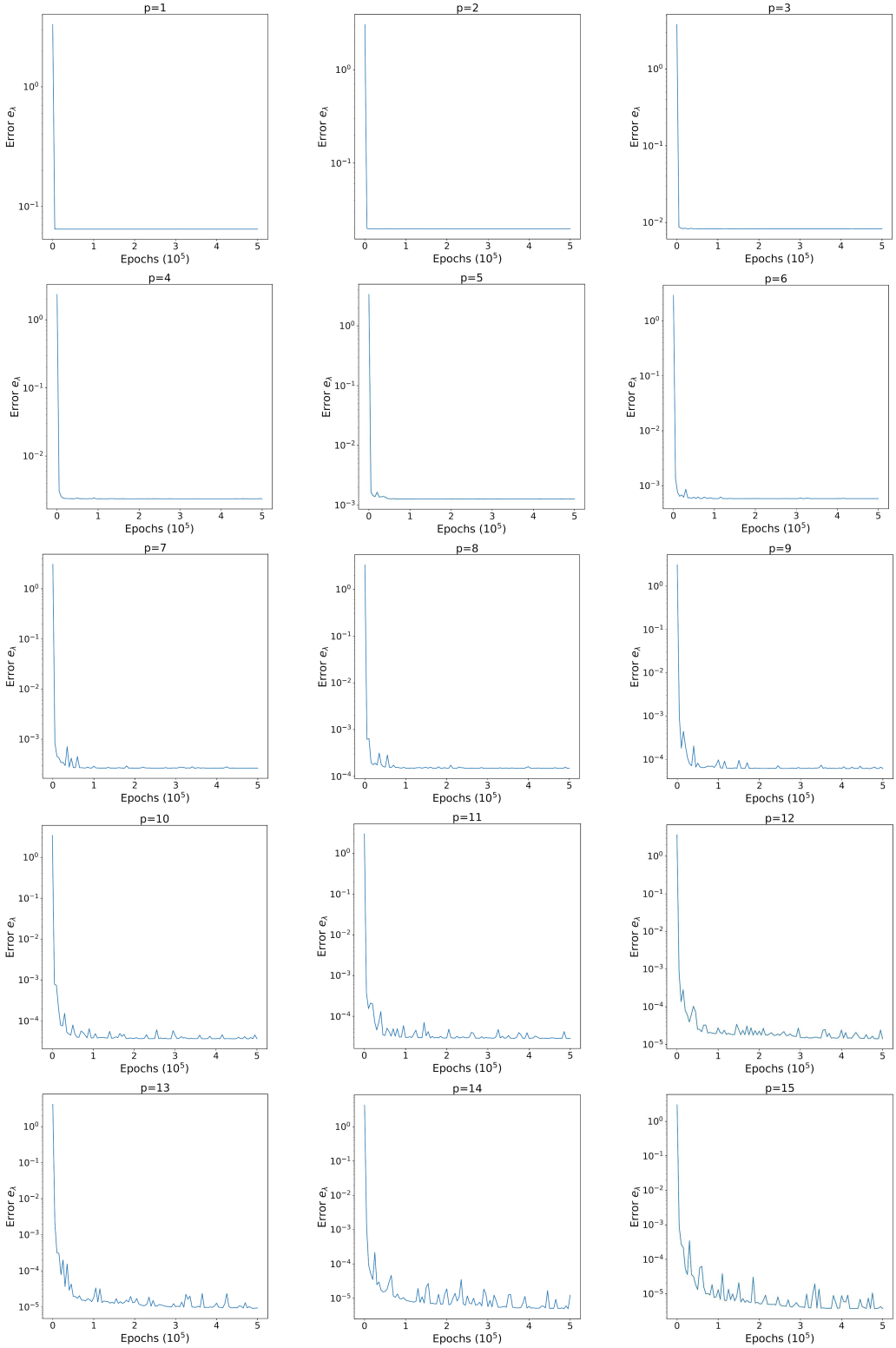

Fig. 5.5. Relative errors during the training process for the coupled harmonic oscillator. The rank p increases from 1 to 30.

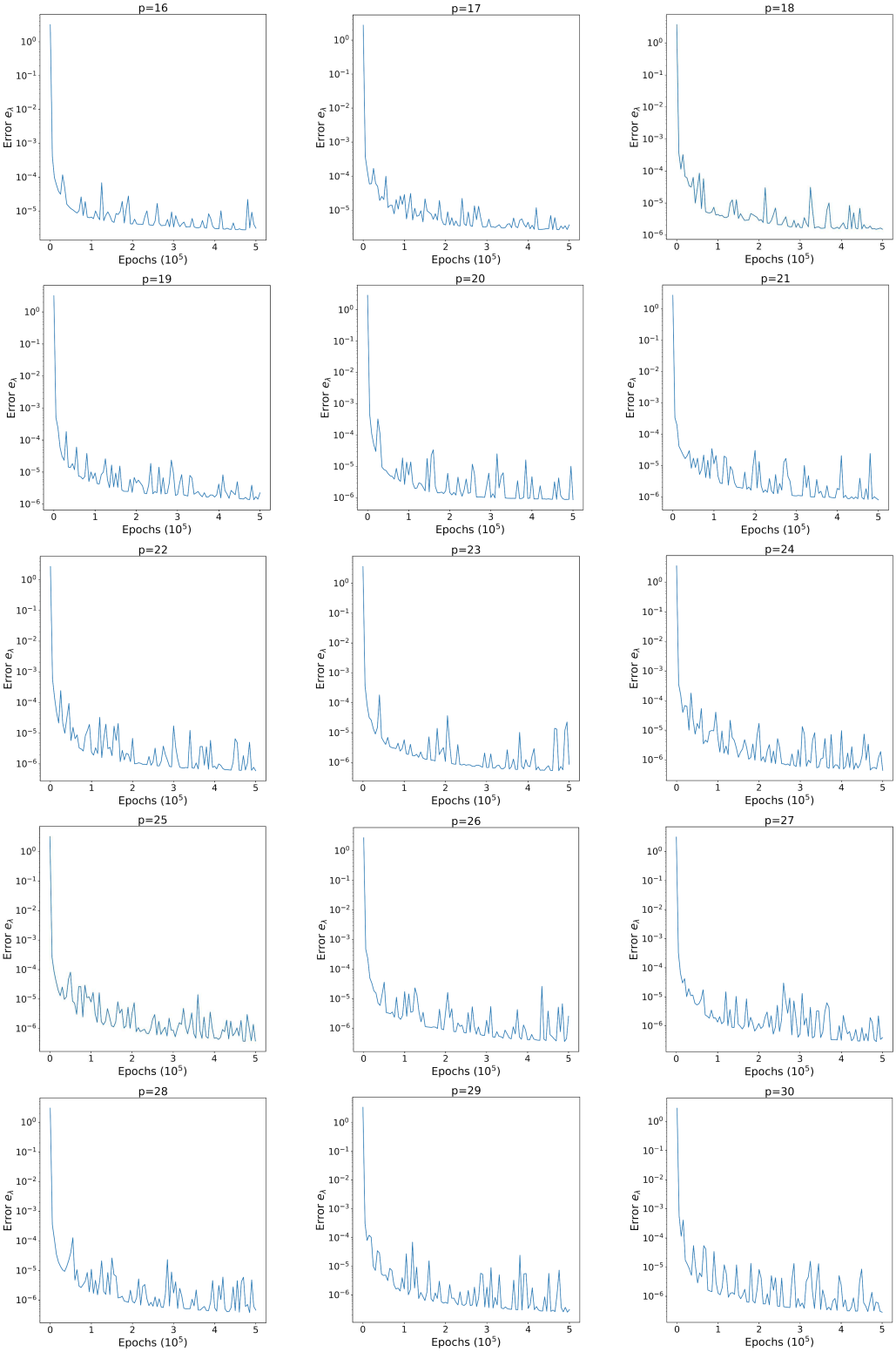

Fig. 5.5. Relative errors during the training process for the coupled harmonic oscillator. The rank p increases from 1 to 30 (cont'd).

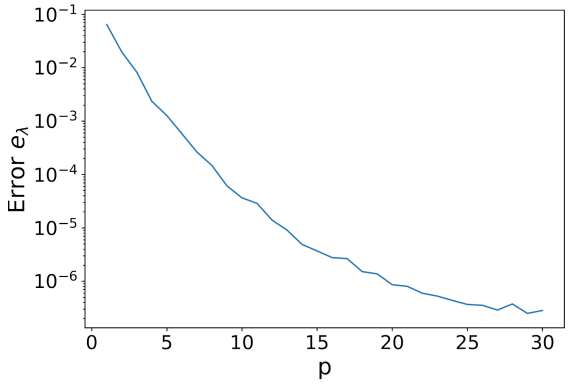

Fig. 5.6. Relative errors e_λ versus hyperparameter p for the coupled harmonic oscillator ($d = 4$).

Since the potential term $1/r_{12}$ in (5.1) can not be expressed as a CP decomposition of finite rank in either Euclidean or spherical coordinates, it is impossible to give the analytical expressions for exact energy E and wave function Ψ and it is also difficult to perform the TNN-based machine learning directly on this potential. Fortunately, Hylleraas [18] chose the three independent variables r_1, r_2, θ, with θ being the angle between r_1 and r_2, to determine the form and the size of a triangle that is composed of the nucleus and two electrons. The coordinates $\{r_1, r_2, \theta\}$ are enough to describe the wave function for the ground state of the helium atom and the corresponding wave function $\Psi(r_1, r_1, \theta)$ satisfies the following eigenvalue problem:

$$-\sum_{i=1}^{2}\frac{1}{2r_i^2}\frac{\partial}{\partial r_i}\left(r_i^2\frac{\partial\Psi}{\partial r_i}\right)-\left(\sum_{i=1}^{2}\frac{1}{2r_i}\right)\cdot\left[\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Psi}{\partial\theta}\right)\right]-\sum_{i=1}^{2}\frac{2}{r_i}\Psi+\frac{1}{r_{12}}\Psi=E\Psi.$$

The volume of this coordinate is $r_1^2 r_2^2 \sin\theta$. The potential $1/r_{12}$ is expanded as functions on the sphere in θ:

$$\frac{1}{r_{12}}=\sum_{\ell=0}^{\infty}\frac{r_<^\ell}{r_>^{\ell+1}}P_\ell(\cos\theta), \tag{5.2}$$

where $r_> = \max\{r_1, r_2\}$ and $r_< = \min\{r_1, r_2\}$, P_ℓ denotes Legendre polynomial of order ℓ. In the implementation, we truncate the expression (5.2) into 20 terms and the computational domain from $[0, +\infty)^2 \times [0, \pi]$ to $[0, 5]^2 \times [0, \pi]$. The benchmark energy for the helium atom is set to be -2.903724377 which is taken from [30], at the level of Born-Oppenheimer nonrelativistic ground state energy. The TNN is set to be $p = 20$. Each subnetwork for the variable r_1, r_2 or θ, respectively, is an FNN with two hidden layers and each hidden layer has 50 hidden neurons. The boundary condition is guaranteed by multiplying the subnetwork in the r_i direction with e^{-r_i}. We train the TNN epochs with a learning rate of 1e-04 in the first 100000 epochs and with a learning rate of 1e-05 in the subsequent 50000 steps to produce the final result. The final energy obtained by the TNN method is -2.903781124 and the relative energy error is 1.9542e-05. Fig. 5.7 shows the radial distribution of electrons for helium atoms. From Fig. 5.7, we know that the TNN method can give good simulations of the real electron distribution.

5.5. Boundary value problem with Neumann boundary condition

In the last example, different from the eigenvalue problems discussed in the above subsections, we tentatively test the performance of TNN for high-dimensional boundary value

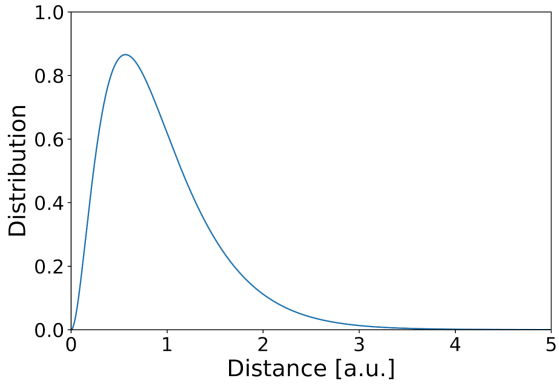

Fig. 5.7. Radial distribution of electrons for helium atom.

problems. For this aim, we consider the following boundary value problem with the Neumann boundary condition:

$$\begin{cases} -\Delta u + \pi^2 u = 2\pi^2 \sum_{i=1}^{d} \cos(\pi x_i), & x \in [0,1]^d, \\ \dfrac{\partial u}{\partial \mathbf{n}} = 0, & x \in \partial[0,1]^d. \end{cases}$$

Then the exact solution is

$$u(x) = \sum_{i=1}^{d} \cos(\pi x_i). \tag{5.3}$$

We use the same loss function as [7] and test cases with $d = 5, 10, 20$, respectively. For all cases, each subnetwork of TNN is an FNN with two hidden layers and each hidden layer has 50 hidden neurons. Referring to the optimization tips in [29], we use the Adam optimizer in the first 100000 steps and then the L-BFGS in the subsequent 50000 steps to produce the final result. The corresponding numerical results for $p = 2d$ are reported in Fig. 5.8 and Table 5.5. The final relative errors have almost the same order of magnitude for different dimensions.

From (5.3), the exact solution can be represented as CP-decomposition with rank d. We can at least claim that the rank of the exact solution is no more than d. For the case $d = 10$, we take hyperparameter p from 1 to 20 and train the TNN with a learning rate of 0.01. Fig. 5.9 shows the final relative errors $\widehat{e}_{L^2}$ and $\widehat{e}_{H^1}$ after 100000 epochs versus p. From Fig. 5.9, we can find an interesting phenomenon that the explicit CP representation (5.3) may not describe the effect of low-rank approximation properly. From (5.3), it looks like the real rank of the exact solution is $p = 10$, but there is no significant error reduction from $p = 5$ to $p = 20$.

Table 5.5: Errors of the Neumann boundary value problem for $d = 5, 10, 20$.

d	e_{L^2}	e_{H^1}
5	4.791e-05	6.079e-04
10	4.520e-05	5.778e-04
20	5.122e-05	6.586e-04

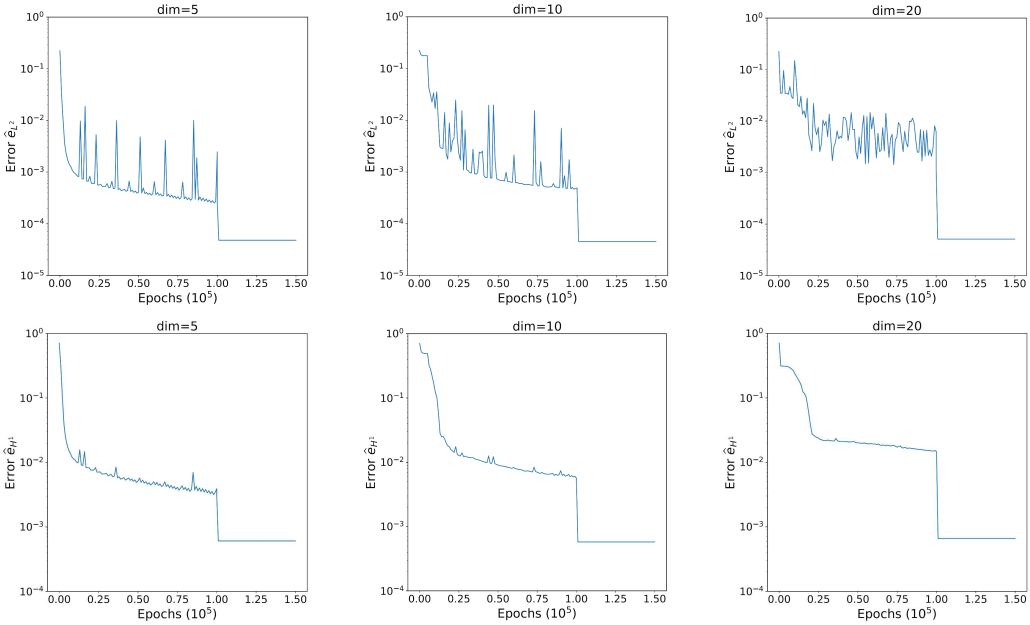

Fig. 5.8. Relative errors during the training process for the Neumann boundary problem for $d = 5, 10, 20$. The top row shows the relative L^2 errors and the bottom one shows the relative H^1 errors of the approximate solution.

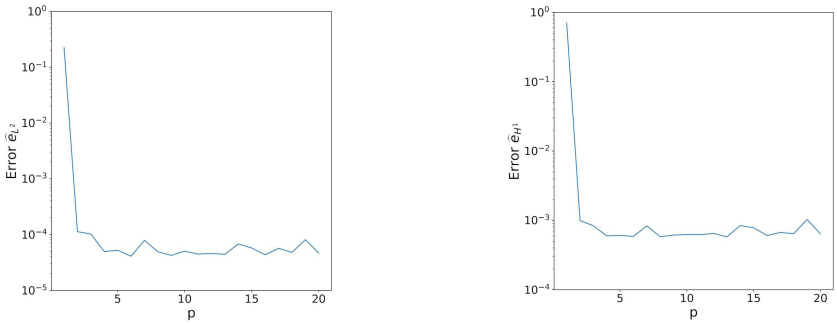

Fig. 5.9. Relative errors versus hyperparameter p for Neumann boundary value problem ($d = 10$). The left subfigure shows the relative L^2 errors and the right one shows the relative H^1 errors of the approximate solution.

6. Conclusions

In this paper, we present the TNN and corresponding machine-learning methods for solving high-dimensional PDEs. Different from the well-known FNN-based machine learning methods, TNN has a tensor product form and its numerical integration can use the fixed quadrature points in each dimension. Benefiting from the tensor product structure, we can design an efficient integration scheme for the functions defined by TNN. These properties lead to TNN-based machine learning that can do the direct inner product computation with the polynomial scale of work for the dimension. We believe that the ability of direct inner production computation will bring more applications in solving high-dimensional PDEs.

Based on the ideas of CP decomposition for tensor product Hilbert space and representing the trial functions by deep neural networks, we introduce the TNN structure, its corresponding approximation property, and an efficient numerical integration scheme. The theoretical results, algorithms, and numerical investigations show that this type of structure has the following advantages:

1. With the help of the straightforward tensor product representation way, we can integrate this type of function separately in the one-dimensional interval. This is the reason that the TNN can overcome the exponential dependence of the computational work for high-dimensional integrations on the dimension.

2. Instead of randomly sampling data points, the training process uses fixed quadrature points. This means that the TNN method can avoid the random sampling process to produce the GD direction in each step and then has better stability.

Besides the above observations, there should exist some interesting topics that need to be addressed in future work:

1. The choice of the subnetwork structure, the activation function, and the more important hyperparameter p.

2. When the computing domain is not tensor-product type, further strategies are demanded to maintain the high efficiency and accuracy of the numerical integration.

3. Since the TNN uses fixed quadrature points, we should design more efficient numerical methods to solve the included optimization problems in the machine learning process.

In addition, more applications to other types of problems should be investigated in the future.

Acknowledgements. This work was supported in part by the National Key Research and Development Program of China (Grant No. 2019YFA0709601), by the National Center for Mathematics and Interdisciplinary Science, CAS.

References

[1] R.A. Adams, *Sobolev Spaces*, Academic Press, 1975.
[2] M. Baymani, S. Effati, H. Niazmand, and A. Kerayechian, Artificial neural network method for solving the Navier-Stokes equations, *Neural Comput. Applic.*, **26**:4 (2015), 765–763.
[3] A. Beygi, S.P. Klevansky, and C.M. Bender, Coupled oscillator systems having partial $\mathcal{PT}$ symmetry, *Phys. Rev. A*, **91**:6 (2015), 062101.
[4] G. Beylkin and M.J. Mohlenkamp, Numerical operator calculus in higher dimensions, *Proc. Natl. Acad. Sci. USA*, **99**:16 (2002), 10246–10251.
[5] G. Beylkin and M.J. Mohlenkamp, Algorithms for numerical analysis in high dimensions, *SIAM J. Sci. Comput.*, **26**:6 (2005), 2133–2159.
[6] W. E, Machine learning and computational mathematics, *Commun. Comput. Phys.*, **28**:5 (2020), 1639–1670.
[7] W. E and B. Yu, The deep Ritz method: A deep-learning based numerical algorithm for solving variational problems, *Commun. Math. Stat.*, **6** (2018), 1–12.
[8] S.W. Ellacott, Aspects of the numerical analysis of neural networks, *Acta Numer.*, **3** (1994), 145–202.

[9] L.C. Evans, *Partial Differential Equations*, AMS, 2010.
[10] M. Griebel and J. Hamaekers, Sparse grids for the Schrödinger equation, *ESAIM Math. Model. Numer. Anal.*, **41** (2007), 215–247.
[11] M. Griebel and S. Knapek, Optimized tensor-product approximation spaces, *Constr. Approx.*, **16** (2000), 525–540.
[12] Y. Gu, C. Wang, and H. Yang, Structure probing neural network deflation, *J. Comput. Phys.*, **434** (2021), 110231.
[13] W. Hackbusch and B.N. Khoromskij, Tensor-product approximation to operators and functions in high dimensions, *J. Complexity*, **23**:4-6 (2007), 697–714.
[14] J. Han, A. Jentzen, and W. E, Solving high-dimensional partial differential equations using deep learning, *arXiv:1707.02568v1*, 2017.
[15] J. Han, L. Zhang, and W. E, Solving many-electron Schrödinger equation using deep neural networks, *J. Comput. Phys.*, **399** (2019), 108929.
[16] J. He, L. Li, J. Xu, and C. Zheng, ReLU deep neural networks and linear finite elements, *J. Comput. Math.*, **38**:3 (2020), 502–527.
[17] D. Hong, T.G. Kolda, and J.A. Duersch, Generalized canonical polyadic tensor decomposition, *SIAM Rev.*, **62**:1 (2020), 133–163.
[18] E.A. Hylleraas, Über den Grundzustand des Heliumatoms, *Z. Physik*, **48** (1928), 469–494.
[19] P. Jin, S. Meng, and L. Lu, MIONet: Learning multiple-input operators via tensor product, *arXiv:2202.06137*, 2022.
[20] D.P. Kingma and J. Ba, Adam: A method for stochastic optimization, *arXiv:1412.6980*, 2014.
[21] S. Knapek, *Hyperbolic Cross Approximation of Integral Operators with Smooth Kernel*, Technical Report 665, SFB 256, University of Bonn, 2000.
[22] S. Knapek, *Approximation und Kompression mit Tensorprodukt-Multiskalenräumen*, Dissertation, Universität Bonn, 2000.
[23] T.G. Kolda and B.W. Bader, Tensor decompositions and applications, *SIAM Rev.*, **51**:3 (2009), 455–500.
[24] I.E. Lagaris, A.C. Likas, and D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Trans. Neural Netw.*, **9**:5 (1998), 987–1000.
[25] I.E. Lagaris, A.C. Likas, and G.D. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, *IEEE Trans. Neural Netw.*, **11**:5 (2000), 1041–1049.
[26] M. Leshno, V.Y. Lin, A. Pinkus, and S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, *Neural Netw.*, **6**:6 (1993), 861–867.
[27] B. Li, S. Tang, and H. Yu, Better approximations of high dimensional smooth functions by deep neural networks with rectified power units, *Commun. Comput. Phys.*, **27** (2020), 379–411.
[28] M.S. Litsarev and I.V. Oseledets, Fast low-rank approximations of multidimensional integrals in ion-atomic collisions modelling, *Numer. Linear Algebra Appl.*, **22**:6 (2015), 1147–1160.
[29] L. Lyu, Z. Zhang, M. Chen, and J. Chen, MIM: A deep mixed residual method for solving high-order partial differential equations, *J. Comput. Phys.*, **452** (2022), 110930.
[30] H. Nakashima and H. Nakatsuji, Solving the Schrödinger equation for helium atom and its isoelectronic ions with the free iterative complement interaction (ICI) method, *J. Chem. Phys.*, **127** (2007), 224104.
[31] M. Raissi, P. Perdikaris, and G.E. Karniadakis, Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations, *arXiv:1711.10561*, 2017.
[32] M.J. Reynolds, A. Doostan, and G. Beylkin, Randomized alternating least squares for canonical tensor decompositions: Application to a PDE with random data, *SIAM J. Sci. Comput.*, **38**:5 (2016), A2634–A2664.
[33] H.J. Schmeisser and H. Triebel, *Topics in Fourier Analysis and Function Spaces*, Wiley, 1987.
[34] J. Shen and H. Yu, Efficient spectral sparse grid methods and applications to high-dimensional elliptic problems, *SIAM J. Sci. Comput.*, **32**:6 (2010), 3228–3250.

[35] J. Shen and H. Yu, Efficient spectral sparse grid methods and applications to high-dimensional elliptic equations II. Unbounded domains, *SIAM J. Sci. Comput.*, **34**:2 (2012), A1141–A1164.

[36] J.W. Siegel and J. Xu, Approximation rates for neural networks with general activation functions, *Neural Netw.*, **128** (2020), 313–321.

[37] J. Sirignano and K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, *J. Comput. Phys.*, **375** (2018), 1339–1364.

[38] Y. Zang, G. Bao, X. Ye, and H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, *J. Comput. Phys.*, **411** (2020), 109409.

[39] D. Zung, The approximation of classes of periodic functions of many variables, *Russian Math. Surveys*, **38** (1983), 117–118.