Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2024-0040

Vol. **38**, No. 3, pp. 603-629
September 2025

Two-Scale Neural Networks for Partial Differential Equations with Small Parameters

Qiao Zhuang[1,†], Chris Ziyi Yao[2], Zhongqiang Zhang[1,*] and George Em Karniadakis[3]

[1] *Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA 01609, USA.*
[2] *Department of Aeronautics, Imperial College London, London, SW7 2AZ, UK.*
[3] *Division of Applied Mathematics, Brown University, Providence, RI 02912, USA. and Pacific Northwest National Laboratory, P.O. Box 999, Richland, 99352, WA, USA.*

Received 27 February 2024; Accepted (in revised version) 8 October 2024

Abstract. We propose a two-scale neural network method for solving partial differential equations (PDEs) with small parameters using physics-informed neural networks (PINNs). We directly incorporate the small parameters into the architecture of neural networks. The proposed method enables solving PDEs with small parameters in a simple fashion, without adding Fourier features or other computationally taxing searches of truncation parameters. Various numerical examples demonstrate reasonable accuracy in capturing features of large derivatives in the solutions caused by small parameters.

AMS subject classifications: 65N35, 35B25

Key words: Two-scale neural networks, partial differential equations, small parameters, successive training.

1 Introduction

In this work, we consider physics-informed neural networks (PINNs) for the following equation

$$p\partial_t u - \epsilon L_2 u + L_0 u = f, \quad \mathbf{x} \in D \subset \mathbb{R}^d, \quad t \in (0,T], \tag{1.1}$$

[†]Current address: School of Science and Engineering, University of Missouri-Kansas City, Kansas City, MO 64110, USA.
[*]Corresponding author. *Email addresses:* `qzhuang@umkc.edu` (Q. Zhuang), `chris.yao20@imperial.ac.uk` (C. Z. Yao), `zzhang7@wpi.edu` (Z. Zhang) `george_karniadakis@brown.edu` (G. E. Karniadakis)

where some proper boundary conditions and initial conditions are imposed. Here $p=0$ or 1, $\epsilon>0$, D is the spatial domain, $d\in\mathbb{N}^+$ and $T>0$. Also, L_2u consists of the leading-order differential operator and L_0u consists of lower-order linear or nonlinear differential operators. For example, consider a singular perturbation problem where $p=0$, $L_2u=\operatorname{div}(a\nabla u)$ is second-order and $L_0u=b\cdot\nabla u+cu$ is first-order.

Small parameters in the equation often pose extra difficulties for numerical methods, see e.g., [29]. The difficulties come from one or more sharp transitions in regions of small volumes, which implies large first-order derivatives or even large high-order derivatives. When $p=0$, $\epsilon>0$ is very small, $L_2u=\Delta u$ and $L_0u=\mathbf{1}\cdot\nabla u$, then at least one boundary layer arises.

When deep feedforward neural networks are used, they are usually trained with stochastic gradient descent methods but do not resolve the issues above as they learn functions with low frequency and small first-order derivatives, see e.g., in [3,28,40].

1.1 Literature review

To deal with functions with high-frequency components such as in singular perturbation problems, at least four approaches have been proposed to address this issue:

- Adding features in the neural networks: Adding random Fourier features is probably the most non-intrusive approach. Adding $\cos(\omega_i^\top x)$,$\sin(\omega_i^\top x)$'s to deep neural networks as approximations of target functions or solutions. With an explicitly specified range for the random frequencies, one can learn a large class of functions. In [5,20,22,33], frequency ranges are scheduled to represent complicated solutions to partial differential equations. See also [21,41,44] for more elliptic type multi-scale PDEs. The Fourier feature networks are also used in [36,37], and see [35] for a review.

- Enhanced loss by adding first-order and higher-order derivatives: Another approach is to include the gradient information in the loss function, e.g., in [6,13,19]. This approach has been applied to solve PDEs, e.g., in [30,42].

- Adaptive weights: In the loss function, adaptive weights are assigned to have a better balance of each squared term in the least-squares formulation, e.g., self-adaptive [26], attention-based weights [2], binary weights [10]. See also [25,39,43].

- Resampling: Sampling points in the loss function can be made adaptive based on the residuals at sampling points during the training process such as in [8,9,23,38]. Also, in [27,31,32], the density function for sampling during the training is computed using the idea of importance sampling. The density function for re-sampling is approximated by Gaussian mixtures and formulated by finding the min-max of the loss function via fixed-point iterations.

Other approaches are not particularly for the high-frequency issue, such as [16] using adaptive activation functions and [34] by respecting causality, continual learning [14] and sequential training [12].

1.2 Related works

One relevant approach to accommodate multiple scales using neural networks is the asymptotic-preserving neural network (APNN) [4, 17, 24]. It involves two-scale expansions and transforming the underlying equation into several equations that may be scale-independent. For example, in [17, 24], the loss function is designed based on equations from a macro-micro decomposition, while in [4], the loss function is rooted in a system in a macroscopic form that corresponds to the governing PDEs. In [11], dimension-augmentation is used to enhance the accuracy, which includes adding the Fourier features. Our approach is closely related to APNN as we explicitly build-in a scale as an extra dimension of input for the neural networks. Another relevant approach is multi-level neural networks [1], addressing the multiple scales in the solution by using a sequence of neural networks with increasing complexity: at each level of the process, it uses a new neural network, to generate a correction corresponding to the error in the previous approximation. Two ingredients are essential in mitigating the effects of high-frequency components: adding Fourier features in the networks and the extra level of refining residuals. A similar approach is in [7] while the scale of residuals is not explicitly incorporated in the solution.

1.3 Contribution and significance of the work

We directly incorporate the scale parameter into the neural networks' architecture, establishing a *two-scale* neural network method. In particular, for a small scale parameter ϵ, the networks employ auxiliary variables related to the scale of ϵ^γ ($\gamma \in \mathbb{R}$) designed to capture intricate features that involve large derivatives such as boundary layers, inner layers, and oscillations. Specifically, we use the feedforward neural network of the form

$$w(\mathbf{x})N(t,\mathbf{x},\epsilon^\gamma(\mathbf{x}-\mathbf{x}_c),\epsilon^\gamma), \quad \gamma<0, \tag{1.2}$$

where $\mathbf{x}_c \in D$. Here $w(\mathbf{x})$ is 1 when boundary layers arise or a simple function satisfying the Dirichlet boundary condition if no boundary layer arises. Without an explicit statement, $\mathbf{x}_c$ is the center of the domain D. To assess the performance of the two-scale neural networks, we will compare their results with those of the following *one-scale* neural networks

$$w(\mathbf{x})N(t,\mathbf{x}). \tag{1.3}$$

We note that when addressing steady-state problems, t is removed from (1.2) and (1.3).

The network (1.2) is motivated as follows. First, it can handle large derivatives for training neural networks. By taking derivatives in x, we readily obtain a factor of ϵ^γ.

Thus, the networks (1.2) can facilitate learning large derivatives or higher frequencies, especially with low-order optimizers. Second, the network (1.2) is analogous to the classical scale expansion of u in ϵ while we do not have explicit expansions. Suppose that $u=\sum_{k=0}^{\infty}u_k\epsilon^{k\beta}$, where $\beta>0$ is problem-dependent. The intuition is that the scale of u_k is about the order of $\epsilon^{-k\beta}$. For the network (1.2), Taylor's expansion in $\mathbf{x}$ leads to a scale expansion in ϵ. Consider the one-dimensional function and the first two derivatives of (1.2). The first derivative is of order ϵ^{γ}. The second derivative is of the order $\epsilon^{2\gamma}$. For singular perturbation problems in this work, β is usually 1/2 or 1. We then choose $\gamma=-1/2$ in all cases. In some cases, $\gamma=-1$ may be a better choice but still can be covered if we use a smooth activation function since $(\epsilon^{-\frac{1}{2}})^2=\epsilon^{-1}$. It may enhance the flexibility of networks by using $\gamma=-1/2$ in all cases, especially when a smooth activation function implies a valid Taylor's expansion in ϵ.

The point $\mathbf{x}_c$ is chosen as the center of the spatial domain D. However, other choices are possible. For example, we can take $\mathbf{x}_c$ to be an endpoint if there is only a boundary layer at one endpoint. However, in most cases, we do not know where large derivatives are. In these cases, we expect that the linear inner layer of the feedforward neural network can help find the best linear combinations of $\epsilon^{\gamma}(\mathbf{x}-\mathbf{x}_c)$ and ϵ^{γ} in training and consequently have locations of large derivatives.

The significance of this work is that the new architecture of networks enables solving problems with small parameters in a simple fashion. For example, we do not need to determine how many equations in APNN, or levels in multilevel neural networks [1], which are problem-dependent. We also avoid the use of Fourier features, a lot of which may be required in two dimensions and even higher dimensions.

We compare our method with multilevel neural networks [1], underscoring our method's capability to capture large derivatives without tuning scaling factors. The design of the two-scale neural networks enables the neural networks to implicitly pick up the scale expansions to accommodate the multiple scales. In contrast, APNN accommodate the multiple scales via explicit asymptotic expansions, resulting in decomposition strategies such as micro-macro and even-odd decomposition. The selection of decomposition strategies and truncation terms of asymptotic expansions is critical to the APNN model's outcomes. Owing to the variability these choices introduce, we refrain from using APNN results as a benchmark to be compared within our study.

The effectiveness of the proposed two-scale approach in handling large derivatives is demonstrated by comprehensive numerical examples in Section 3. These include complex cases such as problems with dual boundary layers in 1D and 2D, viscous Burgers equations, and Helmholtz equations exhibiting oscillations along xy and radial directions, where the corresponding one-scale method falls short in delivering accurate simulations without special treatments.

2 Formulation and algorithms

Under the framework of PINNs, the neural network solution is obtained by minimizing the loss function of residuals. The loss function of the considered problem (1.1) at the continuous level is

$$\int_{[0,T]\times D} r^2(t,x)+\alpha\int_{\partial D}(u-g)^2+\alpha_1\int_D|\nabla u|^2+\beta\cdot\text{loss of initial conditions}, \tag{2.1}$$

where $\alpha\geq 1$, $\alpha_1\geq 0$ and $\beta\geq 0$ and

$$r(t,x)=p\partial_t u-\epsilon L_2 u+L_0 u-f.$$

The discrete formulation corresponding to (2.1) is

$$\frac{1}{N_c}\sum_{i=1}^{N_c}\left|r\left(t_r^i,\boldsymbol{x}_r^i\right)\right|^2+\frac{\alpha}{N_b}\sum_{i=1}^{N_b}\left|u\left(t_b^i,\boldsymbol{x}_b\right)-g^i\right|^2+\frac{\beta}{N_0}\sum_{i=1}^{N_0}\left|u\left(t_0,\boldsymbol{x}_0^i\right)-u_0^i\right|^2+\frac{\alpha_1}{N_c}\sum_{i=1}^{N_c}|\nabla u(t_r^i,\boldsymbol{x}_r^i)|^2, \tag{2.2}$$

where $\{t_r^i,\boldsymbol{x}_r^i\}_{i=1}^{N_c}$ specify the collocation points in the interior time $(0,T]$ and spatial domain D, $\{t_b^i\}$ are time collocation points at $\boldsymbol{x}=\boldsymbol{x}_b$, $\boldsymbol{x}_0^i$ are spacial collocation points at $t=t_0$.

In this paper, we utilize the Adam [18] optimizer to train the neural network parameters. To enhance the accuracy of capturing solutions to problems with small parameters ϵ, we employ a successive training strategy. This approach progressively optimizes the weights and biases of the neural networks, starting with modestly larger ϵ_0 as the initial guess. We summarize the successive training method in Algorithm 1. We remark that this training strategy is similar to continual learning in [14] and sequential training in [12]. However, we fix the size of neural networks, while in [12,14], the network size increases during different stages of training.

3 Numerical results

In this section, we employ the two-scale neural network method for extensive numerical tests on problems where small parameters lead to solutions with large derivatives. The numerical tests cover one-dimensional (1D) ODEs with one (Examples 3.1, 3.6 and 3.7) and two (Example 3.2) boundary layers, 1D viscous Burgers equations with an inner layer (Example 3.3), two-dimensional (2D) steady-state convection-diffusion problems featuring two boundary layers (Example 3.4), and 2D Helmholtz problems characterized by oscillations acting as inner layers (Example 3.5). The numerical results indicate that the two-scale neural network method can provide reasonable accuracy in capturing features arising from large derivatives in solutions. These features include boundary layers, inner layers, and oscillations.

```
Algorithm 1: Successive training of two-scale neural networks for PDEs with
small parameters
  Data: Training set, adaptive learning rates, $\epsilon$ and other parameters from the PDE
  Result: Optimized weights and bias of the neural networks
  Pick $\epsilon_0 = \mathcal{O}(0.1)$ if $\epsilon$ is very small. Initialize the weights and biases randomly.
  Step 1. Solve the PDE with parameter $\epsilon_0$ instead of $\epsilon$. Use the loss function above
    with $\epsilon_0$ in place of $\epsilon$. Train to obtain weights and biases of the neural networks
    (1.2) with $\epsilon = \epsilon_0$.
  Step 2. Pick a positive integer $\ell > 1$. While $\epsilon_0 \geq \ell\epsilon$
            set $\epsilon_0 = \epsilon_0/\ell$ and go to Step 1;
      end
  If $\epsilon_0 < \ell\epsilon$, set $\epsilon_0 = \epsilon$ and go to Step 1. In these iterations, initialize the weights and
    biases obtained from the last iteration.
  Step 3. (Fine-tuning). Use Adam with a smaller learning rate or a second-order
    optimizer to further optimize the weights and biases.
  Step 4. Stop the process if the maximal epoch number is reached.
```

Table 1: Default setup (unless stated otherwise) of the learning rates scheduler, collocation points distribution, and activation function.

Learning Rates (η)	Collocation Points	Activation Function
≤ 10000 steps: $\eta = 10^{-3}$ (1D problems), $\eta = 10^{-2}$ (2D) 10000 to 30000 steps: $\eta = 5\times 10^{-3}$ 30000 to 50000 steps: $\eta = 10^{-3}$ 50000 to 70000 steps: $\eta = 5\times 10^{-4}$ ≥ 70000 steps: $\eta = 10^{-4}$	uniform distribution	tanh

For all the numerical tests, we let $\gamma = -\frac{1}{2}$ in (1.2). Unless stated otherwise, regardless of whether Algorithm 1 is used, we adopt the default setup up of the piecewise constant learning rates scheduler, the uniform distribution for collocation points, and the tanh activation function, as listed in Table 1. All examples are implemented with JAX and the Adam optimizer unless stated otherwise. In some tests, we employ the successive training strategy listed in Algorithm 1. The parameters ϵ_0, ℓ are manually set and are problem-dependent. We let the starting $\epsilon_0 = 0.1$ and $\ell = 10$ unless stated otherwise. In Examples 3.1 and 3.2, when $\epsilon = 10^{-5}$, we use $\ell = 10$ for the successive training until $\epsilon_0 \geq 10^{-4}$ and then $\ell = 2$ to reach $\epsilon = 10^{-5}$.

To further demonstrate the robustness of numerical results, we report the statistical metric $\bar{e} \pm \sigma$ for Examples 3.1 and 3.2. Here, $\bar{e}$ is the average absolute error and σ is the standard deviation.

Example 3.1 (1D ODE with one boundary layer)**.**

$$-\epsilon u''+2u'=3,\quad 0<x<1,\quad u(0)=0,\quad u(1)=0.$$

The solution has a boundary layer at $x=1$. The exact solution to this problem is

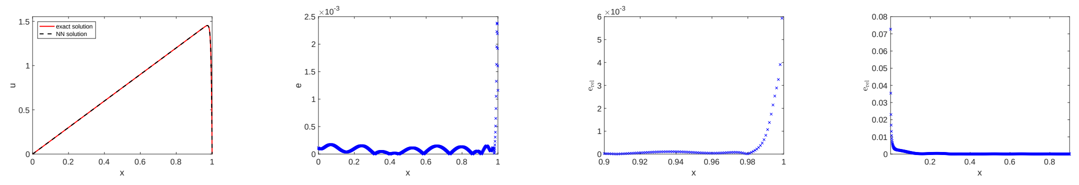
$$u(x)=\frac{3}{2}\left(x-\frac{\exp(-2(1-x)/\epsilon)-\exp(-2/\epsilon)}{1-\exp(-2/\epsilon)}\right).$$

We use the two-scale neural networks $N(x,(x-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon})$ to solve the ODE problem. The neural network (NN) size is $(3,20,20,20,20,1)$. We employ the successive training strategy in Algorithm 1 starting with $\epsilon_0=10^{-1}$, and successively solve the problem for ϵ as small as 10^{-5}, following the sequence $\epsilon_0=10^{-1},10^{-2},10^{-3},10^{-4},5\times10^{-5},2.5\times10^{-5},1.25\times10^{-5},10^{-5}$. The parameters in the discrete loss function in (2.2) and hyper-parameters of the training are summarized in Table 2. We collect the numerical results in Figs. 1 and 5(a) for the case where $\epsilon=10^{-2}$. Observing from Fig. 1(b) to (d), the NN solution matches the exact solution very well, with the relative errors around the boundary layer $x=1$ only at the magnitude of 10^{-3}.

For the case where $\epsilon=10^{-3}$, we collect the results in Figs. 2 and 5(b). As depicted in Fig. 2(a), the NN solution captures the behavior of the exact solution. While there

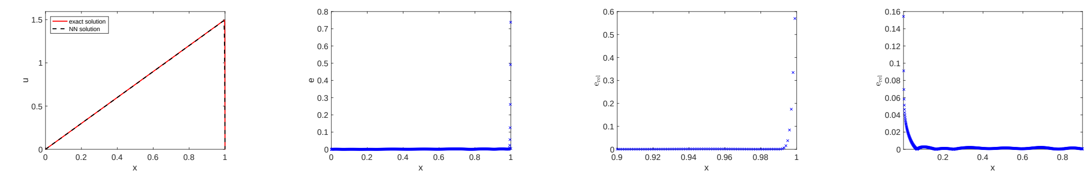
Table 2: Parameters in the loss function (2.2) and hyper-parameters of the successive training for Example 3.1. Here, $\epsilon_s=10^{-5}\times[5,2.5,1.25,1]$, LR is the abbreviation for learning rate, PC is the piecewise constant scheduler in Table 1.

ϵ	10^{-1} (starting ϵ_0)	10^{-2}	10^{-3}	10^{-4}	ϵ_s
α	1	100	1000	1000	1000
α_1	0	0	10^{-6}	10^{-6}	10^{-5}
N_c	200	200	450	450	300
LR	PC	10^{-4}	10^{-4}	10^{-4}	10^{-4}
epochs	20000	30000	50000	50000	30000 each



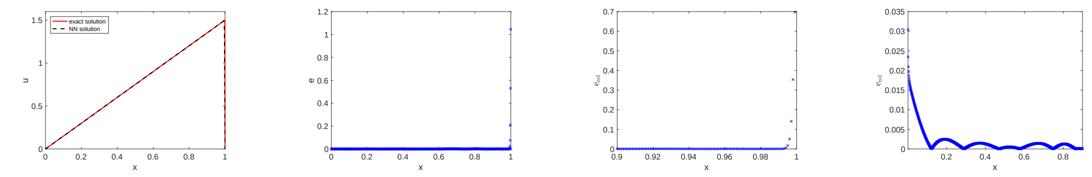
(a) exact and NN solutions (b) absolute error (c) relative error around the boundary layer (d) relative error away from the boundary layer

Figure 1: Numerical results for Example 3.1 when $\epsilon=10^{-2}$ using $N(x,(x-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon})$, with parameters specified in Table 2 .

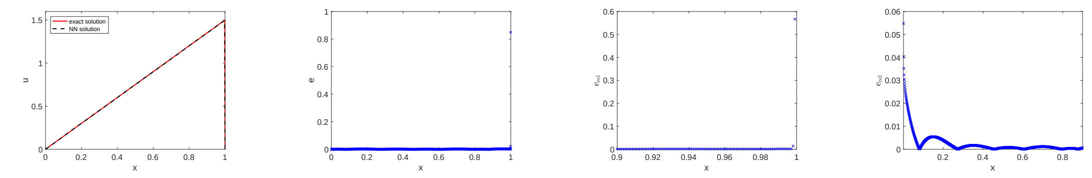


(a) exact and NN solutions (b) absolute error (c) relative error around the boundary layer (d) relative error away from the boundary layer

Figure 2: Numerical results for Example 3.1 when $\epsilon=10^{-3}$, using $N(x,(x-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon})$, with parameters specified in Table 2 .



(a) exact and NN solutions for $\epsilon=10^{-4}$ (b) absolute error (c) relative error around the boundary layer (d) relative error away from the boundary layer



(e) exact and NN solutions for $\epsilon=10^{-5}$ (f) absolute error (g) relative error around the boundary layer (h) relative error away from the boundary layer

Figure 3: Numerical results for Example 3.1 when $\epsilon=10^{-4}$ ((a) to (d)) and 10^{-5} ((e) to (h)), with parameters specified in Table 2.

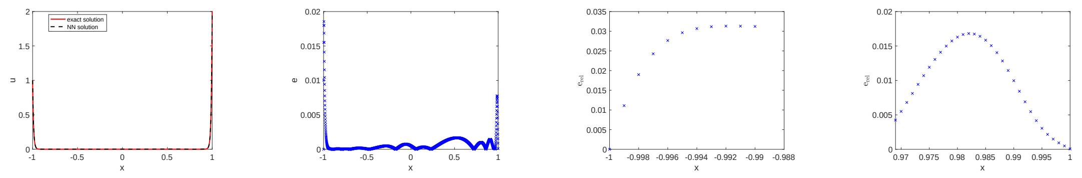
is a relatively larger error observed very close to $x=1$ as shown in Fig. 2(b) and (c), it is essential to note that this challenge is inherent to the nature of the problem itself. In particular, when $\epsilon=10^{-3}$, the boundary layer essentially takes the form of a vertical line. This increases sensitivity to the shift between the actual and predicted boundary layer around $x=1$, as shown in Fig. 2(c). We also present the results for $\epsilon=10^{-4}$ and 10^{-5} in Fig. 3, where we observe comparable accuracy as in the case $\epsilon=10^{-3}$.

Example 3.2 (1D ODE with two boundary layers)**.**

$$\epsilon u''-xu'-u=0, \quad -1<x<1, \quad u(-1)=1, \quad u(1)=2. \tag{3.1}$$

Table 3: Parameters in the loss function (2.2) and hyper-parameters of the successive training for Example 3.2. Here, $\epsilon_s = 10^{-5}\times[5,2.5,1.25,1]$, LR is the abbreviation for learning rate, PC is the piecewise constant scheduler in Table 1.

ϵ	10^{-1} (starting ϵ_0)	10^{-2}	10^{-3}	10^{-4}	ϵ_s
α	1	1	1	1	1
α_1	0	0	0	10^{-4}	10^{-4}
N_c	500	500	430	430	430
LR	PC	PC	10^{-4}	10^{-4}	10^{-4}
epochs	20000	20000	50000	50000	20000 each



(a) exact and NN solutions (b) absolute error (c) relative error around the left boundary layer (d) relative error around the right boundary layer

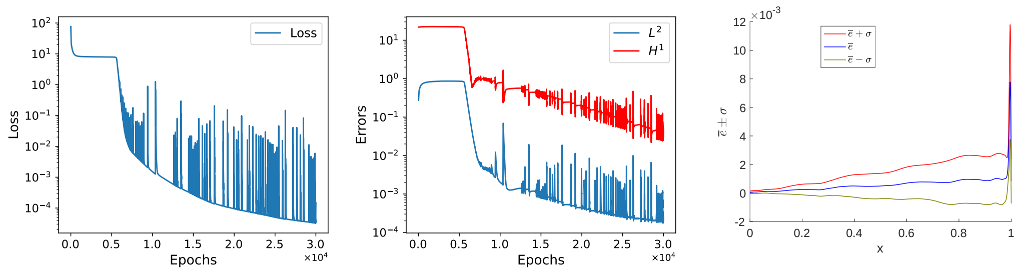
Figure 4: Numerical results for Example 3.2 when $\epsilon = 10^{-2}$ using $N(x, x/\sqrt{\epsilon}, 1/\sqrt{\epsilon})$, with parameters specified in Table 3.

The solution has two boundary layers, at $x = \pm 1$. The exact solution to the problem is

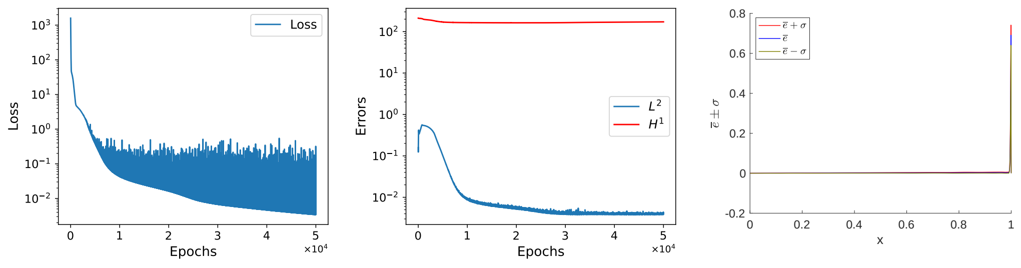
$$\exp\left(\frac{x^2-1}{2\epsilon}\right)\frac{\mathrm{Erf}(\frac{x}{\sqrt{2\epsilon}})+3\mathrm{Erf}(\frac{1}{\sqrt{2\epsilon}})}{2\mathrm{Erf}(\frac{1}{\sqrt{2\epsilon}})}, \quad \mathrm{Erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z \exp(-t^2)\,dt.$$

We solve the considered problem using the two-scale NN $N(x, x/\sqrt{\epsilon}, 1/\sqrt{\epsilon})$. We employ the successive training strategy in Algorithm 1 starting with $\epsilon_0 = 10^{-1}$, and successively solve the problem for ϵ as small as 10^{-5}, following the sequence $\epsilon_0 = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 5\times 10^{-5}, 2.5\times 10^{-5}, 1.25\times 10^{-5}, 10^{-5}$. The parameters in the discrete loss function in (2.2) and hyper-parameters of the training are summarized in Table 3. The NN size is $(3,20,20,20,20,1)$. We collect the numerical results in Figs. 4 and 5(c) for the case when $\epsilon = 10^{-2}$. The figures illustrate the excellent agreement between the NN solution and the exact solution, even at the boundary layers. As shown in Fig. 4 (c) and (d), the relative error is at the level of 10^{-2} in the vicinity of the left and right boundary layers. These results underscore the capability of the two-scale NN method in capturing multiple boundary layers in the exact solution.

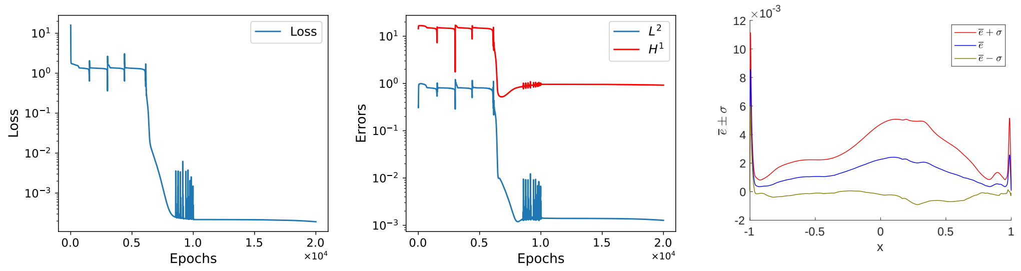
For the scenario when $\epsilon = 10^{-3}$, the boundary layers practically become vertical lines, as shown in Fig. 6(a). We collect the results in Figs. 6 and 5(d). While the NN solution



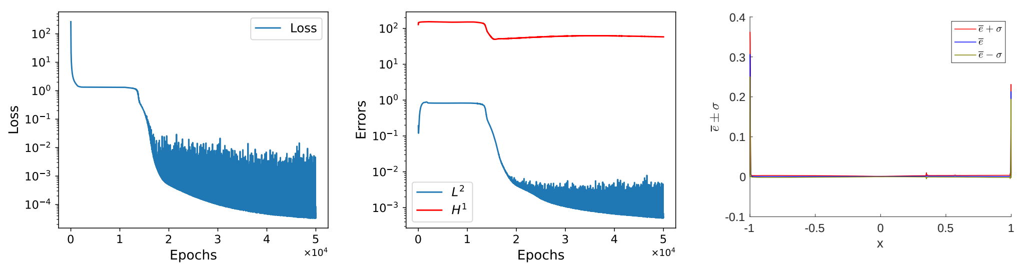
(a) $\epsilon = 10^{-2}$ for Example 3.1



(b) $\epsilon = 10^{-3}$ for Example 3.1

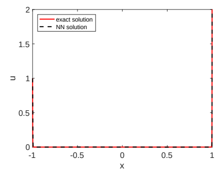


(c) $\epsilon = 10^{-2}$ for Example 3.2

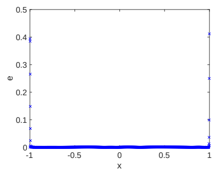


(d) $\epsilon = 10^{-3}$ for Example 3.2

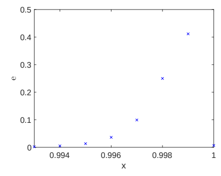
Figure 5: Loss and errors history, as well as the statistical metrics $\bar{e}$ and $\bar{e}\pm\sigma$ for Examples 3.1 and 3.2 when $\epsilon = 10^{-2}$ and $\epsilon = 10^{-3}$.



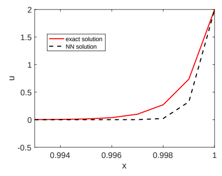
(a) exact and NN solutions



(b) absolute error

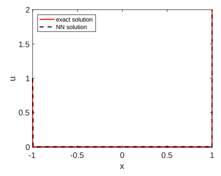


(c) exact and NN solutions around the right boundary

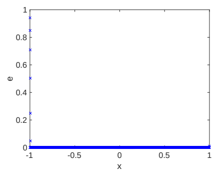


(d) absolute error around the right boundary

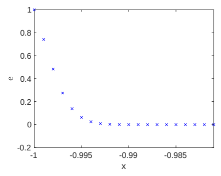
Figure 6: Numerical results for Example 3.2 when $\epsilon=10^{-3}$ using $N(x,x/\sqrt{\epsilon},1/\sqrt{\epsilon})$, with parameters specified in Table 3.



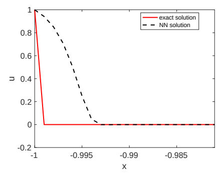
(a) exact and NN solutions



(b) absolute error



(c) exact and NN solutions around the left boundary



(d) absolute error around the left boundary

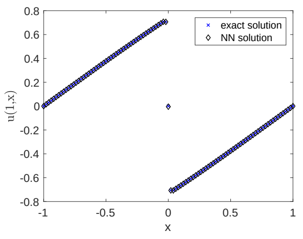
Figure 7: Numerical results for Example 3.2 when $\epsilon=10^{-5}$ using $N(x,x/\sqrt{\epsilon},1/\sqrt{\epsilon})$, with parameters specified in Table 3.

does not achieve the same level of precision as when $\epsilon=10^{-2}$, it is crucial to note that this discrepancy is primarily due to the inherent difficulty of the problem itself: the two boundary layers essentially become vertical lines. Despite the challenge, the two-scale NN solution still captures the key features of the exact solution, as shown in Fig. 6(a). In addition, we provide a visualization of the shift between the actual and predicted solutions in Fig. 6(c) and (d), illustrating that the shift occurs within a very narrow band (width$\approx 4\times 10^{-3}$) around the right boundary. We also present the results for $\epsilon=10^{-5}$ in Fig. 7, where the two-scale NN solution provides comparable accuracy as in the case $\epsilon=10^{-3}$. We closely examine the left boundary, where the largest errors occur, and observe in Fig. 7(d) that the shift between the exact and NN solutions is confined to a very narrow band (width $\approx 10^{-2}$).

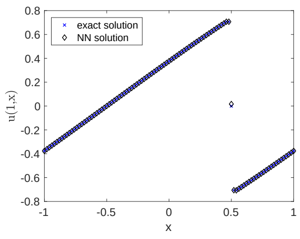
Example 3.3 (1D viscous Burgers' equation)**.**

$$\partial_t u + u u_x - \epsilon u_{xx} = 0 \tag{3.2}$$

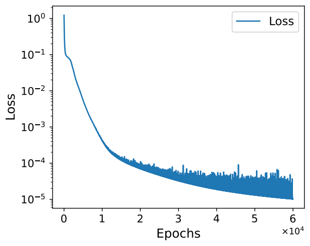
with initial conditions: $u(0,x) = -\sin(\pi(x-x_0))$ and boundary conditions $u(t,-1) =$



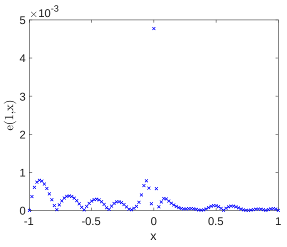
(a) exact and NN solutions when $x_0=0$



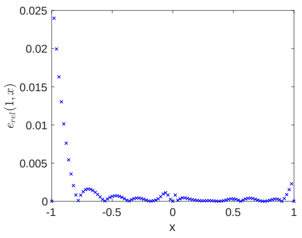
(b) exact and NN solutions when $x_0=0.5$



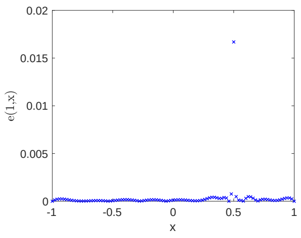
(c) loss history when $x_0=0$



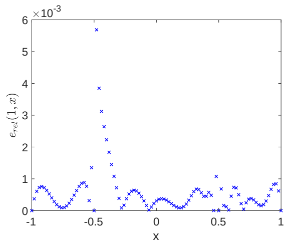
(d) absolute error when $x_0=0$



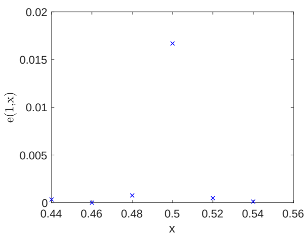
(e) relative error when $x_0=0$



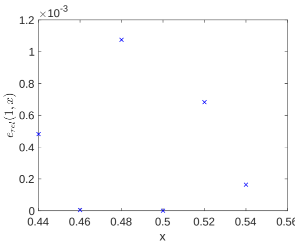
(f) absolute error when $x_0=0.5$



(g) relative error when $x_0=0.5$



(h) absolute error around $x=0.5$ when $x_0=0.5$



(i) relative error around $x=0.5$ when $x_0=0.5$

Figure 8: Numerical results for Example 3.3 when $\epsilon=10^{-2}/\pi$ using $(x^2-1)N(t,x,x/\sqrt{\epsilon},1/\sqrt{\epsilon})+g(t)$ at $t=1$. $\alpha=1,\alpha_1=0$, $N_c=22500$ on the plane (t,x), epochs$=60000$, with constant learning rate 10^{-4} (pre-trained with $\epsilon_0=10^{-1}/\pi$, epochs$=30000$). The relative error is set to zero at $x=-1,0,1$ (when $x_0=0$) and at $x=\pm 0.5$ (when $x_0=0.5$), where $|u(1,x)|<10^{-14}$.

$u(t,1)=g(t)$, where $g(t)$ is determined from the exact solution

$$u(x,t)=-2\epsilon\frac{\partial_x v}{v},\quad \text{where}\quad v=\int_{\mathbb{R}}\exp\left(-\frac{\cos\left(\pi(x-x_0-2\sqrt{\epsilon t}s)\right)}{2\pi\epsilon}\right)e^{-s^2}ds. \tag{3.3}$$

We employ the two-scale NN that enforces the Dirichlet boundary condition, i.e., $(x^2-$

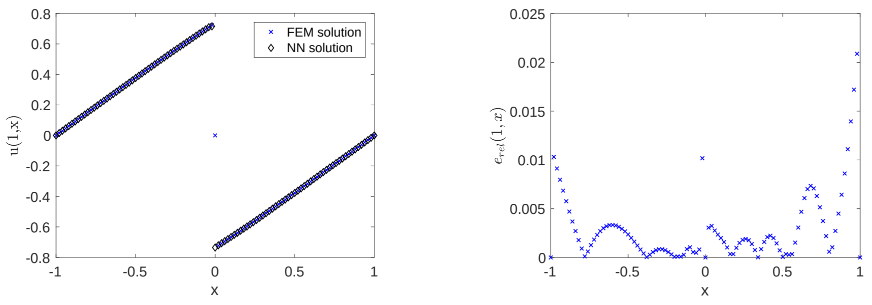


Figure 9: **Left**: high resolution FEM solution with mesh size 1/500, polynomial degree 2 and NN solution obtained with $N(t,x,x/\sqrt{\epsilon},1/\sqrt{\epsilon})$ for Example 3.3 when $\epsilon=10^{-3}/\pi, x_0=0$ at $t=1$. $\alpha=1$, $N_c=22500$ on the plane (t,x), epochs=60000 (two-phase successive training: pre-trained with $\epsilon_0=10^{-2}/\pi$, epochs=60000; the pre-trained parameters are obtained from training with $\epsilon_0=10^{-1}/\pi$, epochs=30000); **right**: relative error.

$1)N(t,x,x/\sqrt{\epsilon},1/\sqrt{\epsilon})+g(t)$ to solve the Burger's equation under a small viscosity parameter ϵ. The NN size is $(4,20,20,20,20,1)$.

First, we discuss the case when $\epsilon=10^{-2}/\pi$. For $x_0=0$, i.e., the inner layer (nearly a shock profile) is located at the center of the domain. According to (3.3), $x_0=0$ leads to $g(t)=0$. We employ the successive training strategy in Algorithm 1 by initializing the parameters of this NN using the pre-trained parameters obtained from training with $\epsilon_0=10^{-1}/\pi$. We collect the numerical results in Fig. 8. From Fig. 8(e), we can see that the maximal relative error at the final time node $t=1$ is 2.5%. The absolute error around the inner layer, i.e., around $x=0$, is only at the level of 10^{-3}, which accounts for a mere 0.4% of the amplitude of the inner layer.

In the scenario where the inner layer is not at the center of the domain, i.e., $x_0=0.5$, we collect the results in Figs. 8(b), (f), (g), (h) and (i), indicating an excellent agreement between the NN and exact solutions. We can see that the maximal relative error at the final time node $t=1$ is within 0.6%, and the absolute error around $x=0.5$ is at the level of 10^{-2}, representing only 1% of the amplitude of the inner layer. The results indicate that the two-scale NN method can accurately capture the exact solution of the Burgers' equation when $\epsilon=10^{-2}/\pi$, regardless of whether the inner layer is near the center of the spatial domain or not.

When $\epsilon=10^{-3}/\pi$, it is difficult to achieve accurate numerical quadrature for the exact solution in (3.3) because the integrand can vary from tiny scales to infinity, driven by the very small ϵ. Therefore, we employ the high-resolution finite element solution as the reference solution. The NN solution is obtained through a two-phase successive training process with Algorithm 1. To be specific, we initialize the NN parameters with the pre-trained parameters obtained from training with $\epsilon_0=10^{-2}/\pi$. These pre-trained parameters themselves were initially obtained from the training process with $\epsilon_0=10^{-1}/\pi$. As observed in Fig. 9, there is a shift between the exact solution and the predicted solution by the two-scale NN method within a very narrow band around the inner layer at $x=0$. The observed deviation primarily arises from the inherent difficulty of the problem itself. Indeed, given $\epsilon=10^{-3}/\pi$, the inner layer practically becomes a vertical line. Nevertheless, the two-scale NN solution still captures the key features of the exact solution in this context.

Example 3.4 (2D steady-state convection-diffusion problem)**.**

$$\begin{aligned} -\epsilon\Delta u+(u_x+u_y)&=f \quad \text{in } \Omega=(0,1)^2,\\ u&=0 \quad \text{on } \partial\Omega. \end{aligned}$$

The exact solution is given by:

$$u=xy\left(1-\exp\left(-\frac{1-x}{\epsilon}\right)\right)\left(1-\exp\left(-\frac{1-y}{\epsilon}\right)\right),$$

and the right-hand side function is then:

$$f=(x+y)\left(1-\exp\left(-\frac{1-x}{\epsilon}\right)\exp\left(-\frac{1-y}{\epsilon}\right)\right)+(x-y)\left(\exp\left(-\frac{1-y}{\epsilon}\right)-\exp\left(-\frac{1-x}{\epsilon}\right)\right).$$

According to [45], the exact solution has two boundary layers around the outflow boundaries $x=1$ and $y=1$. The two-scale NN we employ here is $N(x,(0.5-x)/\sqrt{\epsilon},1/\sqrt{\epsilon},y,(0.5-y)/\sqrt{\epsilon},1/\sqrt{\epsilon}))$, with the size $(6,64,64,64,1)$. The numerical results are collected in Fig. 10 for $\epsilon=10^{-2}$. The figures demonstrate that the NN solution accurately captures the exact solution, including the two boundary layers $x=1$ and $y=1$. Specifically, in Fig. 10(h), we observe that the relative error around these boundary layers remains below 0.8%.

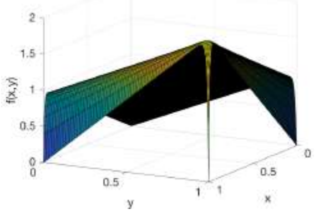
It is important to note that the challenge posed by this problem extends beyond the presence of boundary layers. Another critical aspect lies in the rapid variation of the right-hand-side function f, occurring at $x=1$ and $y=1$, as illustrated in Fig. 10(a). Even so, the prediction by the two-scale NN method is accurate.

Example 3.5 (2D Helmholtz problem)**.** Consider the boundary value problem

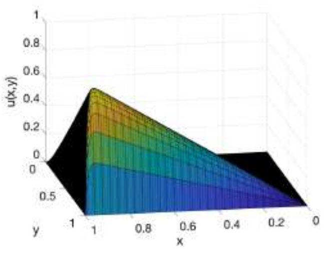
$$\begin{aligned} -\Delta u-k^2u&=f \quad \text{in } \Omega=(-1,1)^2,\\ u&=g \quad \text{on } \partial\Omega. \end{aligned}$$

We consider the following two cases where the exact solutions are respectively given by

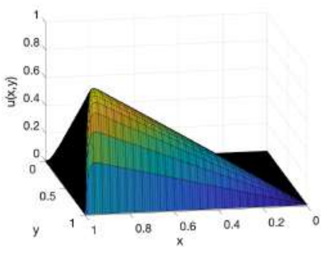
$$\begin{aligned} &\text{a)}\quad u=\sin(a_1\pi x)\sin(a_2\pi y),\\ &\text{b)}\quad u=\frac{\cos(kr)}{k}-\frac{J_0(kr)(J_0(k)\cos k+J_1(k)\sin k)}{k\left(J_0^2(k)+J_1^2(k)\right)}, \end{aligned}$$



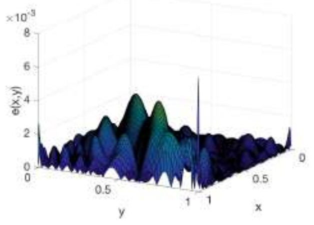
(a) right-hand-side function



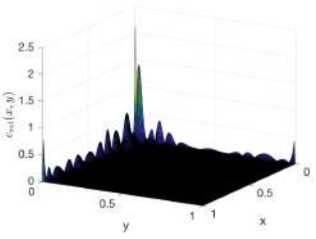
(b) exact solution



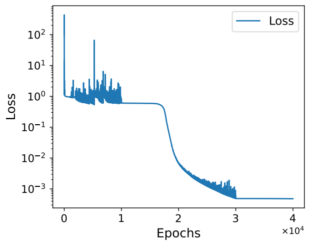
(c) NN solution



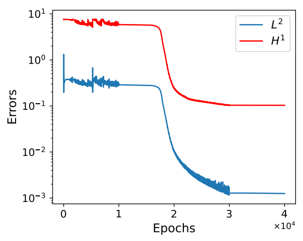
(d) absolute error



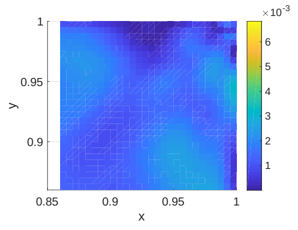
(e) relative error



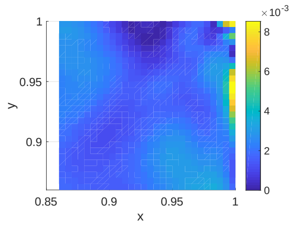
(f) loss history



(g) errors history



(h) absolute error around $x=1$, $y=1$. (top view)

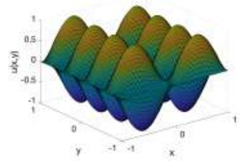


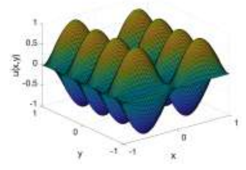
(i) relative error around $x=1$, $y=1$. (top view)

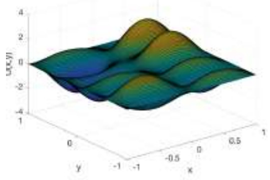
Figure 10: Numerical results for Example 3.4 when $\epsilon=10^{-2}$ using $N(x,(x-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon},y,(y-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon})$ with $\alpha=100$, $\alpha_1=0$, $N_c=22500$, epochs$=40000$. We enforce $e_{rel}(x,y)=0$ on $\partial\Omega$ where the exact solution $u=0$. The relative l^2 error is $e_{rel}^{l^2}=1.5722\times10^{-5}$.

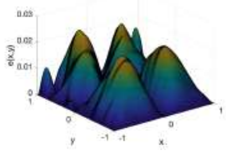
where $r=\sqrt{x^2+y^2}$, $J_\nu(\cdot)$ are Bessel functions of the first kind. These two cases represent the oscillation pattern of the solutions in the x-y direction and the radial direction, respectively.

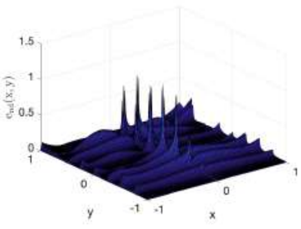
For Example 3.5a), we take $\epsilon=1/\max\{a_1,a_2\}$. In this regard, we address the Helmholtz problem numerically in two scenarios. First, we set $(a_1,a_2)=(1,4)$, which

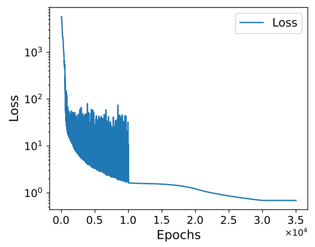

(a) exact solution


(b) two-scale NN solution

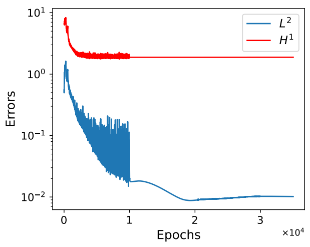

(c) one-scale NN solution


(d) absolute error


(e) relative error



(f) loss history



(g) errors history

Figure 11: **(b),(d),(e),(f),(g) are numerical results with the two-scale NN** for Example 3.5 a), when $k=4$, $a_1=1$, $a_2=4$ using $(x^2-1)(y^2-1)N(x,x/\sqrt{\epsilon},1/\sqrt{\epsilon},y,y/\sqrt{\epsilon},1/\sqrt{\epsilon})$, $\alpha=1$, $\alpha_1=0$, $N_c=22500$, epochs=35000. We enforce $e_{rel}(x,y)=0$ on the nodes where the exact solution $u=0$. The relative l^2 error: $e_{l^2}^{rel}=4.1907\times 10^{-4}$. **(c) is the numerical result with the one-scale NN**, using $(x^2-1)(y^2-1)N(x,y)$, the other configurations are the same as the two-scale method.

corresponds to a modest derivative in the solution and a relatively lower frequency for the time-harmonic wave. Secondly, we consider the case of $(a_1,a_2)=(2,3)$, representing a larger derivative in the solution and a higher frequency for the wave. We employ a two-scale NN framework $(x^2-1)(y^2-1)N(x,x/\sqrt{\epsilon},1/\sqrt{\epsilon},y,y/\sqrt{\epsilon},1/\sqrt{\epsilon})$, where we enforce the homogeneous Dirichlet boundary condition by the construction of the NN. The size of the NN is $(6,10,10,10,10,1)$. The results are collected in Fig. 11. Observing from Fig. 11(a), (b), (d), and (e), the NN solution matches the exact solution well.

As a comparison to the two-scale NN solution, when $(a_1,a_2)=(1,4)$, we present the solution obtained by the one-scale NN, i.e., $(x^2-1)(y^2-1)N(x,y)$ in Fig. 11(c), with NN size $(2,10,10,10,10,1)$. The figure shows that the one-scale NN solution fails to capture the oscillation pattern in the exact solution and deviates significantly from the expected outcome.

We also compare the two-scale NN solutions to those from multi-level neural networks (MLNN) [1], known for their capability to address high-frequency issues. In the multi-level neural network method, a critical technique for addressing high-frequency issues involves scaling the equation's source term with a factor. To be specific, at the initial neural network level, u_0 is solved from the equation with a scaling factor μ_0: $R_0(x,u_0)=\mu_0 f-\mathcal{L}u_0=0$. The NN solution to this equation is denoted as $\tilde{u}_0$, resulting

Table 4: Hyper-parameters used in Example 3.5a) with multi-level neural networks [1].

Hyper-parameters	$\tilde{u}_0$	$\tilde{u}_1$
# Hidden layers	2	2
Widths of each hidden layer	10	20
# Adam iterations	2500	5000
# L-BFGS iterations	200	400
# Wave numbers M	2	4

in the initial approximation $\tilde{u}=\tilde{u}_0/\mu_0$. In the subsequent i-th level ($i\geq 1$) of the neural network, u_i is solved from $R_i(x,u_i)=\mu_i R_{i-1}(x,\tilde{u}_{i-1})-\mathcal{L}u_i=0$. The NN solution for this level is denoted as $\tilde{u}_i$. The updated approximation is then obtained by summing the initial approximation with the cumulative corrections up to the i-th level, i.e.,

$$\tilde{u}=\sum_{i=0}^{L}\frac{\tilde{u}_i}{\prod_{k=0}^{i}\mu_k}.$$

For more details of the algorithm of the multi-level neural networks, we refer the readers to Section 4 of [1].

We specifically investigate the case $(a_1,a_2)=(1,4)$ with the multi-level neural networks. The hyper-parameters used for multi-level NN are summarized in Table 4, which are suggested in [1] for 2D numerical examples. The networks are first trained with Adam, followed by refinement with L-BFGS.

It is recommended in [1] to set $\mu_i,(i\geq 1)$ as the inverse of the amplitude of $u_{E,i}$. Here, $u_{E,i}$ is a rough approximation of the solution $\hat{u}_i$ to $\mathcal{L}\hat{u}_i-R_{i-1}=0$ by the extreme learning machine (ELM) [15].

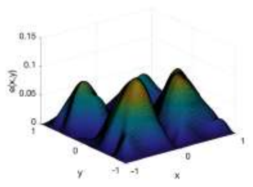
For $\tilde{u}_i$, it is of the form:

$$\sin(\boldsymbol{\omega_M}\pi(1-x))\sin(\boldsymbol{\omega_M}\pi(1-y))N(\gamma(x),\gamma(y)), \tag{3.4}$$

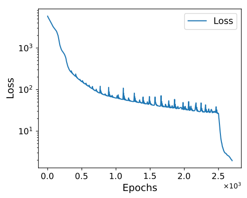
where the Fourier feature: $\gamma(x)=[\cos(\boldsymbol{\omega_M}x),\sin(\boldsymbol{\omega_M}x)]$, and $\boldsymbol{\omega_M}=(\omega_1,\cdots,\omega_M)$, M is specified in Table 4.

We compare the two-scale NN solution to the MLNN's initial and first levels of approximated solutions: $\tilde{u}_0/\mu_0$ and $\tilde{u}_0/\mu_0+\tilde{u}_1/\mu_0\mu_1$, with the hyper-parameters in Table 4. The two-scale NN results are obtained using the NN size of (3,10,1), trained for 2500 epochs with Adam, followed by 200 epochs using L-BFGS. The MLNN results are obtained both without and with the Fourier feature in (3.4). We denote the absolute error between the exact solution and the i-th level approximated solution as $e_i=\left|u-\sum_{k=0}^{i}\left(\tilde{u}_k/\prod_{j=0}^{k}\mu_j\right)\right|$. The loss history is scaled by dividing $(\prod_{k=0}^{i}\mu_k)^2$, where i denotes the level at which the current loss function values are being calculated.

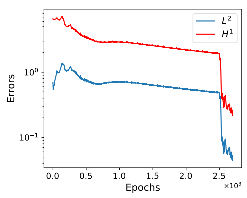
As we can see in Fig. 12(a) and (d) (e), the predictions by the two-scale NN and MLNN without Fourier feature provide reasonable accuracy in approximating the exact solution. The figure shows that the initial level prediction from the MLNN is on par with the



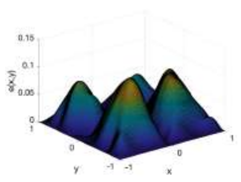
(a) absolute error of two-scale NN solution



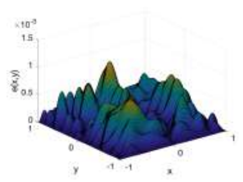
(b) loss history



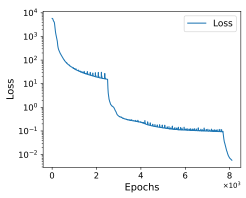
(c) errors history



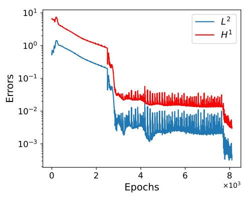
(d) e_0 with MLNN without Fourier feature



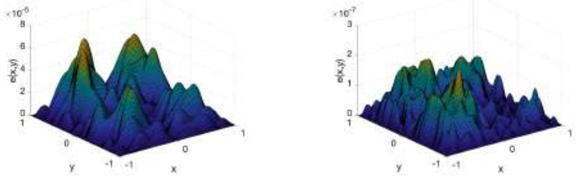
(e) e_1 with MLNN without Fourier feature



(f) loss history



(g) errors history



(h) e_0 (left) and e_1 (right) with MLNN, adding Fourier feature

Figure 12: **(a)-(c)**: Results for Example 3.5a) with two-scale NN: $(x^2-1)(y^2-1)N(x,x/\sqrt{\epsilon},1/\sqrt{\epsilon},y,y/\sqrt{\epsilon},1\sqrt{\epsilon})$, when $k=4$, $a_1=1$, $a_2=4$, with $\alpha=1$, $\alpha_1=0$, $N_c=10000$. **(d)-(g)**: MLNN with NN: $(x^2-1)(y^2-1)N(x,y)$; **(h):** MLNN with Fourier feature in (3.4). Hyper-parameters are specified in Table 4.

two-scale NN when trained with the same number of epochs. The MLNN's first-level correction refines the accuracy to 10^{-3}. The results of the MLNN with the Fourier feature (3.4) shown in Fig. 12(h) indicate that adding the Fourier feature significantly improves the accuracy.

The results suggest that in addressing high-frequency issues, the two-scale NN does not necessarily surpass the approach of adding Fourier features or MLNN in accuracy; however, it offers a simpler solution by avoiding special treatments such as selecting wave numbers for Fourier features or scaling factors for MLNN. To further compare the two-scale NN method with MLNN, we consider two more 1D problems with one boundary layer, to be presented in Examples 3.6 and 3.7.

For Example 3.5b), to avoid evaluating singular derivatives of the solution, we ex-

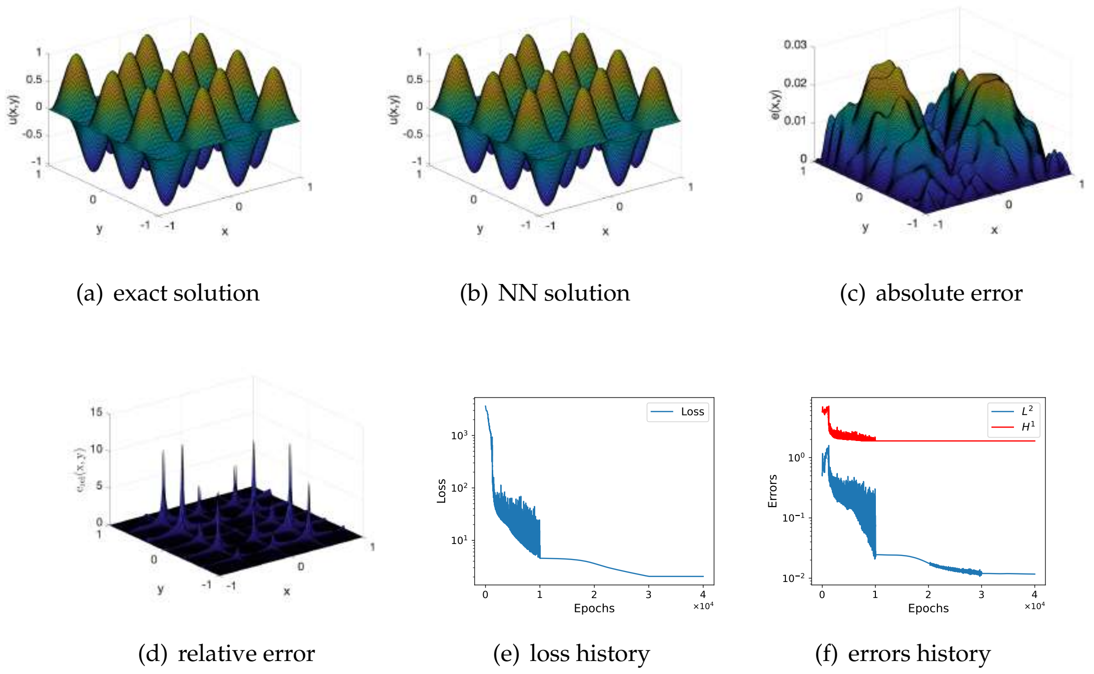


Figure 13: Numerical results for Example 3.5a), when $k=3$, $a_1=2$, $a_2=3$ using $(x^2-1)(y^2-1)N(x,x/\sqrt{\epsilon},1/\sqrt{\epsilon},y,y/\sqrt{\epsilon},1/\sqrt{\epsilon})$, $\alpha=1$, $\alpha_1=0$, $N_c=22500$, epochs$=40000$. We enforce $e_{rel}(x,y)=0$ on the nodes where the exact solution $u=0$. The relative l^2 error: $e^{rel}_{l^2}=5.5318\times 10^{-4}$.

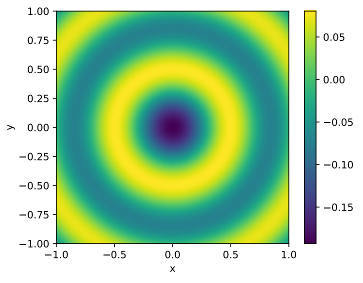
clude the sampling points close to the origin. To utilize NN solutions that can exactly enforce the Dirichlet boundary condition for the problem, we consider the following auxiliary problem:

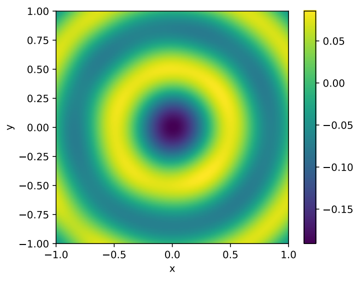
$$\begin{aligned} -\Delta\tilde{u}-k^2\tilde{u}&=f \quad \text{in } \Omega=(-1,1)^2, \\ \tilde{u}&=0 \quad \text{on } \partial\Omega. \end{aligned} \tag{3.5}$$

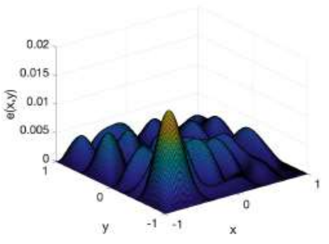
Then $u=\tilde{u}+g_b$, where g_b is defined to be

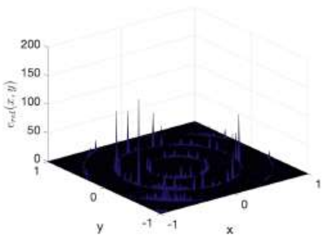
$$g_b(x,y)=x^2(g_2(y)-g(1,1))+y^2(g_1(x)-g(1,1))+g(1,1),$$

with $g_2(y)=g(1,y)$, $g_1(x)=g(x,1)$. Taking $\epsilon=1/k^2$, we utilize the neural networks $(x^2-1)(y^2-1)N(x,x/\sqrt{\epsilon},y,1/\sqrt{\epsilon},y/\sqrt{\epsilon},1/\sqrt{\epsilon})$ to solve the auxiliary problem (3.5) and, consequently, obtain the solution u to the original Helmholtz problem. We collect the numerical results in Fig. 14. The figure demonstrates that the two-scale NN solution accurately captures the radial oscillation pattern for a modest wave number $k=10$, an achievement the corresponding one-scale approach does not replicate under the same configuration.


(a) exact solution


(b) NN solution


(c) absolute error


(d) relative error

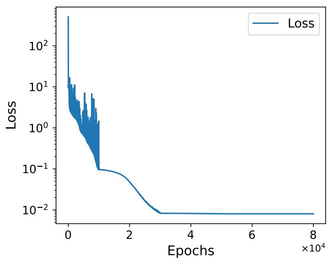

(e) loss history

Figure 14: Numerical results for Example 3.5b) when $k=10$ using $(x^2-1)(y^2-1)N(x,x/\sqrt{\epsilon},y,1/\sqrt{\epsilon},y/\sqrt{\epsilon},1/\sqrt{\epsilon})$ with $k=1/\sqrt{\epsilon}$, $\alpha=1$, $\alpha_1=0$, $N_c=22497$, epochs= 80000. The relative l^2 error is: $e_{l^2}^{rel}=3.41\times10^{-3}$.

Example 3.6 (1D ODE with one boundary layer, comparison with multi-level neural networks)**.**

$$-\varepsilon u''+u'=1,\qquad u(0)=0,\quad u(1)=0.$$

The example is the same as Example 5.2 in [1].

We compare the results of solving this problem using the two-scale NN method and MLNN. To generate the MLNN results, we replicate the neural network configuration with the Fourier feature as described in Example 5.2 of [1]:

$$\sin(\boldsymbol{\omega_M}\pi(1-x))N(\gamma(x)),\tag{3.6}$$

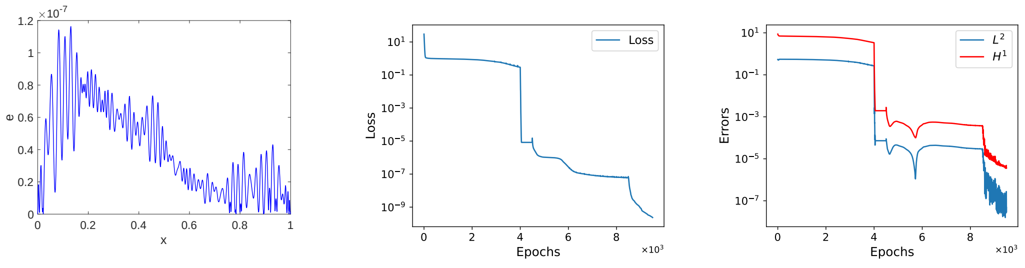
where $\gamma(x)=[\cos(\boldsymbol{\omega_M}x),\sin(\boldsymbol{\omega_M}x)]$, $\boldsymbol{\omega_M}=(\omega_1,\cdots,\omega_M)$. We denote the absolute error between the exact solution and the i-th level approximated solution as $e_i=\left|u-\sum_{k=0}^{i}\left(\tilde{u}_k/\prod_{j=0}^{k}\mu_j\right)\right|$.

The MLNN results are obtained with the hyper-parameters listed in Table 5 and presented in Figs. 15 and 16.

For the case $\epsilon=10^{-2}$, as shown in Fig. 15, even though we use only one level of correction, the accuracy of the approximation achieves 10^{-7} in the absolute error. The

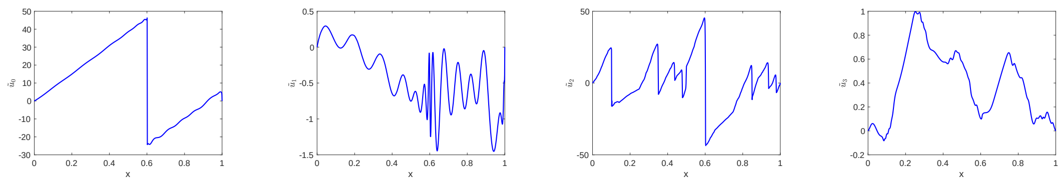
Table 5: Hyper-parameters used in Example 3.6 with multi-level neural networks [1].

Hyper-parameters	$\tilde{u}_0$	$\tilde{u}_1$	$\tilde{u}_2$	$\tilde{u}_3$
# Hidden layers	1	1	1	1
Width of each hidden layer	5	10	20	20
# Adam iterations	4000	4000	4000	10000
# L-BFGS iterations	500	1000	2000	0
# Wave numbers M	3	5	7	3

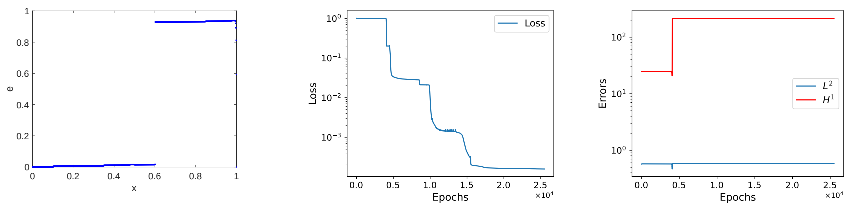


(a) absolute error e_1 (b) loss history (c) errors history

Figure 15: Numerical results for Example 3.6 when $\epsilon=10^{-2}$, using MLNN adding the Fourier feature in (3.6), with one level of correction.

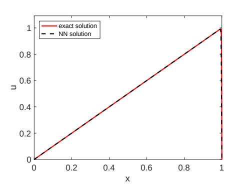


(a) $\tilde{u}_0$ (b) $\tilde{u}_1$ (c) $\tilde{u}_2$ (d) $\tilde{u}_3$

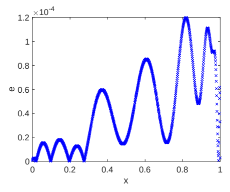


(e) absolute error e_3 (f) loss history (g) errors history

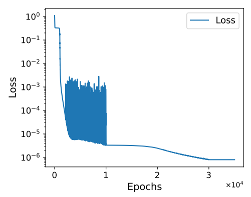
Figure 16: Numerical results for Example 3.6 when $\epsilon=10^{-3}$, using MLNN adding the Fourier feature in (3.6), with three levels of correction.



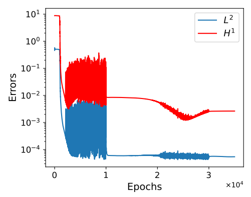
(a) exact and NN solutions for $\epsilon = 10^{-2}$



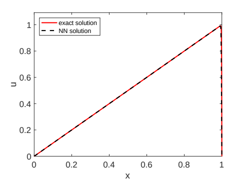
(b) absolute error



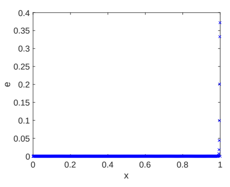
(c) loss history



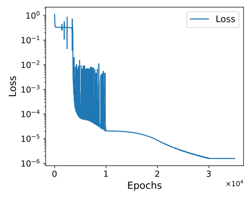
(d) errors history



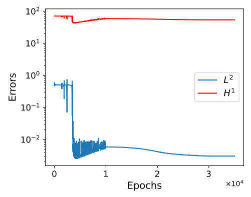
(e) exact and NN solutions for $\epsilon = 10^{-3}$



(f) absolute error



(g) loss history



(h) errors history

Figure 17: Numerical results for Example 3.6 using $N(x,(x-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon})$: **(a)-(d)**: when $\epsilon = 10^{-2}$, with $\alpha = 1$, $\alpha_1 = 0$, $N_c = 200$, epochs$=35000$; **(e)-(h)**: when $\epsilon = 10^{-3}$, with $\alpha = 1$, $\alpha_1 = 0$, $N_c = 450$, epochs$=35000$.

loss and error histories demonstrate fast convergence and are consistent with Figure 17 in [1]. The accuracy can be further refined to 10^{-9} if three levels of corrections are applied.

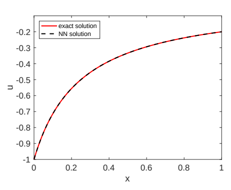
However, for the case $\epsilon = 10^{-3}$, as shown in Fig. 16, the MLNN results have not captured the exact solution, despite the employment of three levels of corrections.

As a comparison, we present the results using the two-scale NN method in Fig. 17, with the NN size of $(3,20,20,20,20,1)$ and hyper-parameters in Table 1.

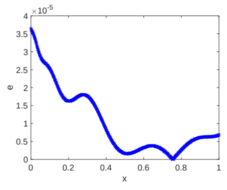
When $\epsilon = 10^{-2}$, as illustrated in Fig. 17(a) to (d), the absolute error reaches the magnitude of 10^{-4} after 35000 iterations. This is less precise compared to the MLNN results, where an accuracy of the order of 10^{-7} is achieved with just one level of correction.

When $\epsilon = 10^{-3}$, we solve the equation directly without employing the successive training strategy in Algorithm 1. As depicted in Fig. 17(e) to (h), the two-scale NN method successfully captures the key features of the exact solution. The overall accuracy is similar to that shown in Fig. 2 for Example 3.1 (a similar 1D problem with one boundary layer).

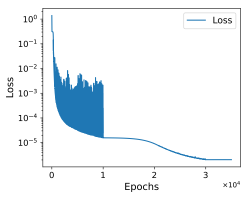
In summary, this case study demonstrates that although the two-scale NN method may not necessarily surpass the MLNN method in terms of accuracy, it offers a simple approach to solving PDEs with small parameters. Furthermore, as the parameter in the PDE becomes smaller, the two-scale NN method can still provide reasonable accuracy with only a little or no special treatment.



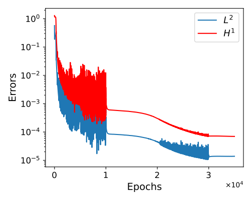
(a) exact and NN solutions for $\epsilon=1/8$



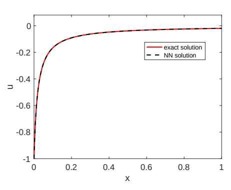
(b) absolute error



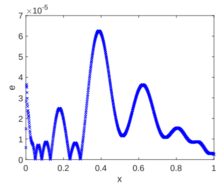
(c) loss history



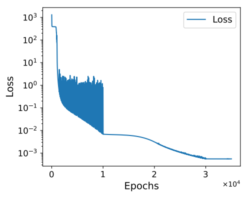
(d) errors history



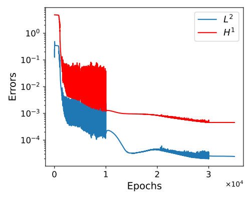
(e) exact and NN solutions for $\epsilon=10^{-2}$



(f) absolute error



(g) loss history



(h) errors history

Figure 18: Numerical results for Example 3.7 using $N(x,(x-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon})$: **(a)-(d)**: when $\epsilon=1/8$ $(k=8)$, with $\alpha=1$, $\alpha_1=0$, $N_c=200$, epochs$=35000$; **(e)-(h)**: when $\epsilon=10^{-2}$ $(k=100)$, with $\alpha=1000$, $\alpha_1=0$, $N_c=450$, epochs$=35000$.

Example 3.7 (1D steady-state Burgers' equation)**.**

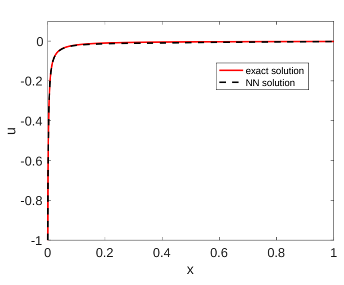
$$u''-kuu'=0,\quad 0<x<1,\quad u(0)=-1,\quad u(1)=-2/(k+2).$$

This example, a nonlinear 1D ODE with a boundary layer, is the same as Example 7 in [1]. We solve this problem with the two-scale NN method with the NN size of (3,20,20,20,20,1) and discuss the results alongside those presented in Example 7 of [1] using MLNN. Here, we take $\epsilon=1/k$.

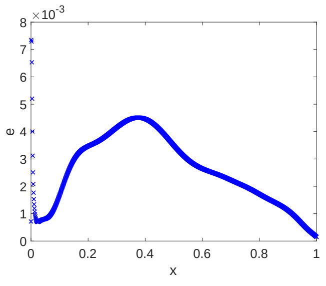
When $k=8$ ($\epsilon=1/8$, a larger ϵ), as shown in Fig. 18(a) to (d), the accuracy of the two-scale NN method reaches 10^{-5}. In contrast, the MLNN results for this example, as reported in [1], achieve an accuracy of 10^{-13} with three levels of correction. Although the two-scale NN results do not reach the accuracy of those provided by MLNN, it is worth mentioning that the two-scale NN results are obtained without adding Fourier features and are optimized with only a first-order optimizer (Adam).

When ϵ is modestly small, such as $\epsilon=10^{-2}$ ($k=100$), as shown in Fig. 18(e) to (h), the two-scale NN method can directly handle it without any special treatment, achieving an accuracy of 10^{-5}. As ϵ decreases further, for example to $\epsilon=10^{-3}$ ($k=1000$), the successive training strategy in Algorithm 1 can be utilized to enhance the training outcomes. As demonstrated in Fig. 19, the two-scale NN method achieves an accuracy of 10^{-3} for $\epsilon=10^{-3}$ by employing Algorithm 1 with starting $\epsilon_0=10^{-2}$ and $\ell=2$.

Similar to our observations in Example 3.6, this case study shows that while the two-



(a) exact and NN solutions



(b) absolute error

Figure 19: Numerical results for Example 3.7 when $\epsilon = 10^{-3}$ using $N(x,(x-0.5)/\sqrt{\epsilon},1/\sqrt{\epsilon})$ and Algorithm 1, with parameters specified in Table 6.

Table 6: Parameters in the loss function (2.2) and hyper-parameters for the successive training for Example 3.7, LR is the abbreviation for learning rate, PC is the piecewise constant scheduler in Table 1.

ϵ	10^{-2} (starting ϵ_0)	5×10^{-3}	2.5×10^{-3}	1.25×10^{-3}	10^{-3}
α	1000	1000	1000	10000	10000
α_1	0	0	0	0	0
N_c	450	450	450	450	450
LR	PC	10^{-4}	10^{-4}	10^{-4}	10^{-4}
epochs	35000	35000	35000	35000	35000

scale NN method may not necessarily exceed the accuracy of the MLNN method, it provides a simple approach to solving PDEs with small parameters. When the parameter in the PDE is modestly small, the two-scale NN method can deliver reasonable accuracy without any special treatment. For significantly smaller values of ϵ, the successive training strategy can be employed to facilitate reasonably accurate outcomes.

4 Conclusion and discussion

In this work, we construct the two-scale neural networks by explicitly incorporating small parameters in PDEs into the architecture of feedforward neural networks. The construction enables solving problems with small parameters simply, without modifying the formulations of PINNs or adding Fourier features in the networks. To enhance the accuracy of neural network predictions, especially for PDEs with smaller parameters, a successive training strategy is introduced. Extensive numerical tests illustrate that the two-scale neural network method is effective and provides reasonable accuracy in capturing features associated with large derivatives in solutions. These features include boundary layers, inner layers, and oscillations.

Notably, when a parameter in the PDEs becomes extremely small, the method struggles to accurately capture large derivatives. However, for problems with extremely small parameters, the proposed two-scale neural network method may be employed as a good initial guess for further training.

Acknowledgments

We thank Professor Serge Prudhomme and Mr. Ziad Aldirany from Polytechnique Montréal for providing the multi-level neural networks code for Examples 1 and 5.2 in their paper [1] and Example 3.5 a) in this work. We also thank Professor Houman Owhadi from the California Institute of Technology for helpful discussions. GEK acknowledges support from the DOE SEA-CROGS project (DE-SC0023191) and the MURI-AFOSR project (FA9550-20-1-0358).

References

[1] Z. Aldirany, R. Cottereau, M. Laforest, and S. Prudhomme. Multi-level neural networks for accurate solutions of boundary-value problems. *Comput. Methods Appl. Mech. Engrg.*, 419:116666, 2024.

[2] S. J. Anagnostopoulos, J. D. Toscano, N. Stergiopulos, and G. E. Karniadakis. Residual-based attention and connection to information bottleneck theory in PINNs. *Comput. Methods Appl. Mech. Engrg.*, 421:116805, 2024.

[3] R. Basri, M. Galun, A. Geifman, D. Jacobs, Y. Kasten, and S. Kritchman. Frequency bias in neural networks for input of non-uniform density. *arXiv:2003.04560*, 2020.

[4] G. Bertaglia, C. Lu, L. Pareschi, and X. Zhu. Asymptotic-preserving neural networks for multiscale hyperbolic models of epidemic spread. *Math. Models Methods Appl. Sci.*, 32(10):1949–1985, 2022.

[5] W. Cai, X. Li, and L. Liu. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM J. Sci. Comput.*, 42(5):A3285–A3312, 2020.

[6] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu. Sobolev training for neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Adv. Neural Inf. Process. Syst.*, volume 30. Curran Associates, Inc., 2017.

[7] Z. Fang, S. Wang, and P. Perdikaris. Ensemble learning for physics informed neural networks: A gradient boosting approach. *arXiv:2302.13143*, 2023.

[8] Z. Gao, T. Tang, L. Yan, and T. Zhou. Failure-informed adaptive sampling for PINNs, part II: Combining with re-sampling and subset simulation. *arXiv preprint arXiv:2302.01529*, 2023.

[9] Z. Gao, L. Yan, and T. Zhou. Failure-informed adaptive sampling for PINNs. *SIAM J. Sci. Comput.*, 45(4):A1971–A1994, 2023.

[10] Y. Gu, H. Yang, and C. Zhou. SelectNet: self-paced learning for high-dimensional partial differential equations. *J. Comput. Phys.*, 441:Paper No. 110444, 18, 2021.

[11] W. Guan, K. Yang, Y. Chen, S. Liao, and Z. Guan. A dimension-augmented physics-informed neural network (DaPINN) with high level accuracy and efficiency. *J. Comput. Phys.*, 491:Paper No. 112360, 2023.

[12] J. Han and Y. Lee. Hierarchical learning to solve pdes using physics-informed neural networks. In *Computational Science – ICCS 2023: 23rd International Conference, Prague, Czech Republic, July 3–5, 2023, Proceedings, Part III*, page 548–562, Berlin, Heidelberg, 2023. Springer-Verlag.

[13] J. Hoffman, D. A. Roberts, and S. Yaida. Robust learning with Jacobian regularization, 2020.

[14] A. A. Howard, S. H. Murphy, S. E. Ahmed, and P. Stinis. Stacked networks improve physics-informed training: Applications to neural networks and deep operator networks. *arXiv preprint arXiv:2311.06483*, 2023.

[15] G.-B. Huang, D. Wang, and Y. Lan. Extreme learning machines: A survey. *Int. J. Mach. Learn. Cybern*, 2:107–122, 06 2011.

[16] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J. Comput. Phys.*, 404:109136, 2020.

[17] S. Jin, Z. Ma, and K. Wu. Asymptotic-preserving neural networks for multiscale time-dependent linear transport equations. *J. Sci. Comput.*, 94(3):Paper No. 57, 21, 2023.

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[19] S. Lämmle, C. Bogoclu, K. Cremanns, and D. Roos. Gradient and uncertainty enhanced sequential sampling for global fit. *Comput. Methods Appl. Mech. Engrg.*, 415:Paper No. 116226, 23, 2023.

[20] X.-A. Li, Z.-Q. J. Xu, and L. Zhang. A multi-scale DNN algorithm for nonlinear elliptic equations with multiple scales. *Commun. Comput. Phys.*, 28(5):1886–1906, 2020.

[21] X.-A. Li, Z.-Q. J. Xu, and L. Zhang. Subspace decomposition based DNN algorithm for elliptic type multi-scale PDEs. *J. Comput. Phys.*, 488:Paper No. 112242, 17, 2023.

[22] Z. Liu, W. Cai, and Z.-Q. J. Xu. Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains. *Commun. Comput. Phys.*, 28(5):1970–2001, 2020.

[23] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[24] Y. Lu, L. Wang, and W. Xu. Solving multiscale steady radiative transfer equation using neural networks with uniform stability. *Res. Math. Sci.*, 9(3):45, 2022.

[25] S. Maddu, D. Sturm, C. L. Müller, and I. F. Sbalzarini. Inverse dirichlet weighting enables reliable training of physics informed neural networks. *Mach. Learn.: Sci. Technol.*, 3(1):015026, 2022.

[26] L. D. McClenny and U. M. Braga-Neto. Self-adaptive physics-informed neural networks. *J. Comput. Phys.*, 474:Paper No. 111722, 23, 2023.

[27] M. A. Nabian, R. J. Gladstone, and H. Meidani. Efficient training of physics-informed neural networks via importance sampling. *CoRR*, abs/2104.12325, 2021.

[28] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310, 2019.

[29] H.-G. Roos, M. Stynes, and L. Tobiska. *Robust Numerical Methods for Singularly Perturbed Differential Equations*. Springer, 2008.

[30] H. Son, J. W. Jang, W. J. Han, and H. J. Hwang. Sobolev training for physics informed neural networks. *Commun. Math. Sci.*, 21(6):1679–1705, 2023.

[31] K. Tang, X. Wan, and C. Yang. DAS-PINNs: A deep adaptive sampling method for solving high-dimensional partial differential equations. *J. Comput. Phys.*, 476:Paper No. 111868, 26, 2023.

[32] K. Tang, J. Zhai, X. Wan, and C. Yang. Adversarial adaptive sampling: Unify PINN and optimal transport for the approximation of PDEs. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

[33] B. Wang, W. Zhang, and W. Cai. Multi-scale deep neural network (MscaleDNN) methods for oscillatory Stokes flows in complex domains. *Commun. Comput. Phys.*, 28(5):2139–2157, 2020.

[34] S. Wang, S. Sankaran, and P. Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *Comput. Methods Appl. Mech. Engrg.*, 421:116813, 2024.

[35] S. Wang, S. Sankaran, H. Wang, and P. Perdikaris. An expert's guide to training physics-informed neural networks. *arXiv:2308.08468*, 2023.

[36] S. Wang, H. Wang, and P. Perdikaris. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Comput. Methods Appl. Mech. Engrg.*, 384:Paper No. 113938, 28, 2021.

[37] S. Wang, H. Wang, and P. Perdikaris. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Comput. Methods Appl. Mech. Engrg.*, 384:113938, 2021.

[38] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Comput. Methods Appl. Mech. Engrg.*, 403:115671, 2023.

[39] Z. Xiang, W. Peng, X. Liu, and W. Yao. Self-adaptive loss balanced physics-informed neural networks. *Neurocomputing*, 496:11–34, 2022.

[40] Z.-Q. J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *Commun. Comput. Phys*, 28(5):1746–1767, 2020.

[41] A. L. Yang and F. Gu. A mesh-less, ray-based deep neural network method for the Helmholtz equation with high frequency. *Int. J. Numer. Anal. Model.*, 19(4):587–601, 2022.

[42] J. Yu, L. Lu, X. Meng, and G. E. Karniadakis. Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems. *Comput. Methods Appl. Mech. Engrg.*, 393:Paper No. 114823, 22, 2022.

[43] G. Zhang, H. Yang, F. Zhu, Y. Chen, et al. Dasa-Pinns: Differentiable adversarial self-adaptive pointwise weighting scheme for physics-informed neural networks. *SSRN*, 2023.

[44] R. Zhang. Learning high frequency data via the coupled frequency predictor-corrector triangular DNN. *Jpn. J. Ind. Appl. Math.*, 40(2):1259–1285, 2023.

[45] Z. Zhang. Finite element superconvergence on Shishkin mesh for 2-D convection-diffusion problems. *Math. Comp.*, 72(243):1147–1177, 2003.