A Tailored Physics-informed Neural Network Method for Solving Singularly Perturbed Differential Equations

Yiwen Pang
College of Computer Science and Technology/Artificial Intelligence, Nanjing University of Aeronautics and Astronautics
Nanjing, China
yiwenpang@nuaa.edu.cn

Ye Li
College of Computer Science and Technology/Artificial Intelligence, Nanjing University of Aeronautics and Astronautics, MIIT Key Laboratory of Pattern Analysis and Machine Intelligence
Nanjing, China
yeli20@nuaa.edu.cn

Shengjun Huang
College of Computer Science and Technology/Artificial Intelligence, Nanjing University of Aeronautics and Astronautics, MIIT Key Laboratory of Pattern Analysis and Machine Intelligence
Nanjing, China
huangsj@nuaa.edu.cn

ABSTRACT

Physics-informed neural networks (PINNs) have recently been demonstrated to be effective for the numerical solution of differential equations, with the advantage of small real labelled data needed. However, the performance of PINN greatly depends on the differential equation. The solution of singularly perturbed differential equations (SPDEs) usually contains a boundary layer, which makes it difficult for PINN to approximate the solution of SPDEs. In this paper, we analyse the reasons for the failure of PINN in solving SPDE and provide a feasible solution by adding prior knowledge of the boundary layer to the neural network. The new method is called the tailored physics-informed neural network (TPINN) since the network is tailored to some particular properties of the problem. Numerical experiments show that our method can effectively improve both the training speed and accuracy of neural networks.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

physics-informed neural network, singularly perturbed differential equation, boundary layer, prior knowledge

ACM Reference Format:
Yiwen Pang, Ye Li, and Shengjun Huang. 2022. A Tailored Physics-informed Neural Network Method for Solving Singularly Perturbed Differential Equations. In *2022 5th International Conference on Algorithms, Computing and Artificial Intelligence (ACAI 2022), December 23–25, 2022, Sanya, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3579654.3579674

1 INTRODUCTION

Differential equations generally describe systems in the fields of applied physics and applied mathematics [1]. Singularly perturbed differential equations (SPDEs) are differential equations (ordinary or partial) that depend on a small positive parameter ε, and whose solutions (or their derivatives) approach a discontinuous limit as ε approaches zero. SPDEs have been successfully applied to many fields, such as fluid dynamics and chemical reactions [2]. Generally, these differential equations are difficult to solve analytically. The solution to SPDEs is much harder since it contains a very small parameter ε.

Traditionally, people focused on the asymptotic expansion method [3] and the numerical methods such as the finite difference method [4, 5] and finite element method [6, 7] to find the approximate solutions of SPDEs. However, these methods are either complicated in calculation or expensive in computation. With the rapid growth of deep learning in recent years, using neural networks to represent the approximate solution of various differential equations has attracted the attention of many researchers. [8] first used an artificial neural network to solve ordinary and partial differential equations. After that, many scholars solved differential equations by using different neural networks, including feedforward neural networks [9], radial basis function neural networks [10], finite element neural networks [11], and Legendre neural networks [12]. Among them, physics-informed neural networks (PINNs) are one of the most popular methods to solve differential equations.

[13] proposed the pioneering work of PINNs, to solve both forward and inverse problems involving nonlinear PDEs. PINN is a kind of neural network that embeds differential equations into neural network training. It is a data-driven method to approximate the solutions of very complex differential equations that we cannot solve precisely. The traditional PINN puts the basic information of the differential equations into the objective function so that the actual data collected from the model are replaced because this knowledge is already contained in the formula. On the one hand, the size of the dataset is largely decreased, so we do not have to collect data. On the other hand, the model we train will not be overfitted with the embedding of differential equations. PINN has demonstrated remarkable power in applications including fluid dynamics [14–16], biomedical engineering [17], meta-material design [18, 19], software packages [20], and numerical simulators [21, 22]. Recently, many variants of PINN have also been developed to improve its performance. A Lagrangian physics-informed neural network is designed for computational fluid dynamics [23]. A parareal physics-informed neural network is designed for time-dependent PDEs [24].

ACAI 2022, December 23–25, 2022, Sanya, China

ACM ISBN 978-1-4503-9833-6/22/12…$15.00
https://doi.org/10.1145/3579654.3579674

These PINN variants are aimed at some special differential equations. A Bayesian physics-informed neural network is designed for noisy data [25]. A hybrid PINN based on a convolutional network uses a convolution kernel instead of automatic differentiation [26]. A variational physics-informed neural network based on the Galerkin method is used to speed up training in the complex domain [27]. These PINN variants optimize the network structure or calculation to improve network performance.

For the SPDEs, the small positive coefficient ε will make some terms in the differential equation vanish. This causes PINN to stop iterative training because PINN mainly depends on the information of the equation. In this paper, we propose a tailored physics-informed neural network (TPINN) to solve SPDEs by adding extra structure information about the solution. The extra information is splitting the solution into two parts: smooth and steep. Usually, the steep part is called layer in this case. With this kind of solution splitting technique, our TPINN method can behave better than the traditional PINN, and our TPINN method is stable with a small positive parameter ε.

2 ANALYZING THE FAILURE CAUSE OF PINN SOLVING SPDE

2.1 Physics-informed neural networks

Considering the following form of PDE [13],

$$\begin{aligned} \mathcal{D}(u(\mathbf{x})) &= f(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \\ u(\mathbf{x}_b) &= B, \qquad \mathbf{x}_b \in \partial\Omega. \end{aligned} \tag{1}$$

Here, $\mathcal{D}$ is the differential operator, and equation $u(\mathbf{x}_b) = B$ is the boundary condition. For such a partial differential equation given the definite conditions, we can determine its unique solution $u(\mathbf{x})$. PINN uses a neural network $NN(\mathbf{x})$ to approximate the solution $u(\mathbf{x})$. To make NN satisfy the above formula, we put the differential equation information into the objective function of NN:

$$L(\mathbf{x};\theta) = MSE(\mathcal{D}(NN(\mathbf{x})) - f(\mathbf{x})) + MSE(NN(\mathbf{x}_b) - B), \tag{2}$$

where MSE is the common mean square error, and θ are the parameters of the neural network NN. Then, we use an optimization algorithm to optimize $L(\theta)$.

$$\theta^* = \arg\min_{\theta} L(\mathbf{x};\theta). \tag{3}$$

2.2 Singularly perturbed differential equations

Assuming that a singularly perturbed differential equation (SPDE) is given in the form [2],

$$\begin{aligned} -\varepsilon y^{(n)} + L(y^{(n-1)}, y^{(n-2)}, ..., y, x) &= f(x), \qquad \varepsilon > 0, x \in \Omega, \\ y(x_b) &= B, \qquad x_b \in \partial\Omega. \end{aligned} \tag{4}$$

In this paper, we set L a linear operator and focus on $n = 2$. ε is a positive parameter and we set $\varepsilon \to 0$. ε is located at the highest derivative of the differential equation, and the solution of the equation will be greatly affected by ε. See Fig.1 for an example. When ε decreases, the highest derivative collapses. As a result, the solution of this differential equation has already changed and usually contains a special structure called layers.

2.3 Why PINN fails to solve SPDE

In this subsection, we analyse why PINN may fail to solve SPDE. Let us consider a first-order singularly perturbed differential equation:

$$\begin{aligned} -\varepsilon y' + (2x+1)y &= -2x^3 - x^2 + 2\varepsilon x, \qquad \varepsilon > 0, x \in [0,1], \\ y(0) &= e^{-\frac{2}{\varepsilon}}. \end{aligned} \tag{5}$$

The actual solution of the equation is $y = e^{\frac{x^2+x-2}{\varepsilon}} - x^2$. The image of the solution is shown in Fig.1. It can be seen from the structure of the solution that term $e^{\frac{x^2+x-2}{\varepsilon}}$ is the solution of the homogeneous version of Eq.(5) and the term $-x^2$ is a special solution of Eq.(5). This means that $-x^2$ also meets Eq.(5). When ε decreases, term $e^{\frac{x^2+x-2}{\varepsilon}}$ will fade away, and the initial condition gradually becomes $y(0) = 0$. This makes the neural network assume that $y = -x^2$ is the actual solution to the equation. In fact, term $e^{\frac{x^2+x-2}{\varepsilon}}$ has not disappeared. The equation will give confusing information to PINN and the outcome is shown in Fig.1.

The reason for this failure is that neural networks have difficulty fitting the boundary layer. We can guide the training direction of the neural network by adding additional information about the boundary layer.

3 TAILORED PHYSICS-INFORMED NEURAL NETWORKS

In this section, we present our TPINN method in detail. In order to add more information about the solution, we need to understand its structure. It is obvious that only a few kinds of differential equations have analytical solutions. However, can still obtain the basic structure of the solution of the linear differential equation. First let us consider a simple convection-diffusion equation [2]

$$\begin{aligned} -\varepsilon u'' + b(x)u' + c(x)u &= f(x) \quad \text{for } x \in (0,1), \\ u(0) &= u(1) = 0, \\ c(x) &\geq 0, \text{ for } x \in [0,1]. \end{aligned} \tag{6}$$

We define the transformation $\xi = \frac{1-x}{\varepsilon}$. Theoretically, the expansion of the solution u_{as} is [2]

$$\begin{aligned} u_{as}(x) &= u_g(x) + v_{loc}(\xi) \\ &= \sum_{\nu=0}^{m} \varepsilon^{\nu} u_{\nu}(x) + \sum_{\mu=0}^{m} \varepsilon^{\mu} v_{\mu}(\frac{1-x}{\varepsilon}), \\ |u(x) - u_{as}(x)| &\leq C\varepsilon^{m+1} \quad \text{for } x \in [0,1] \text{ and } \varepsilon \leq \varepsilon_0. \end{aligned} \tag{7}$$

Where $u_g(x)$ is the global expansion, representing most of the area of the solution (excluding the boundary layer), $v_{loc}(\xi)$ represents the local expansion, which is the correction of the solution in the boundary layer area, where v_μ should satisfy the boundary layer equation. Usually the structure of the boundary layer depends on the boundary layer equation. The simplest type of boundary layer is the exponential boundary layer (decaying exponential function), also known as the ordinary boundary layer.

The boundary layer shown by Eq.(7) is obviously at the boundary $x = 1$. To design of the neural network, we need to add the circumstances that there may be layers at all boundaries. We plan to fit v_μ using multiple small networks. For Eq.(6), we can design

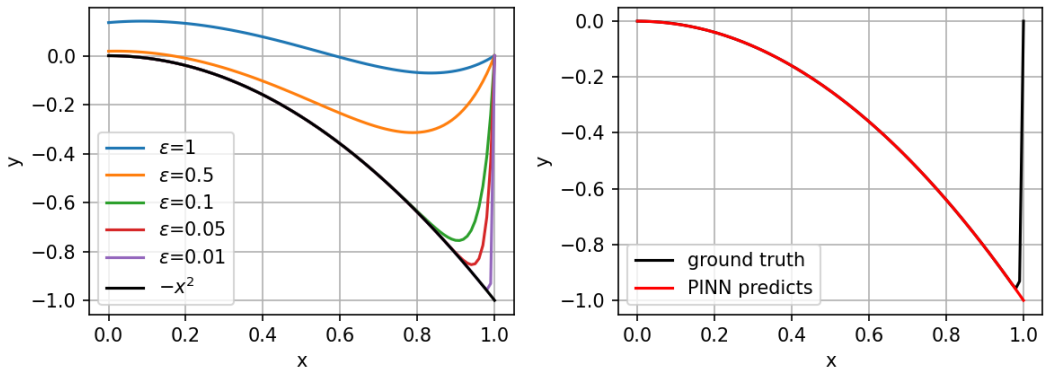

Figure 1: Left: The actual solution of Eq.(5) when ε reduces. Right: The red and black lines are the PINN fit outcome and actual solution of Eq.(5) when $\varepsilon = 0.01$.

the whole model be like

$$NN_{new}(x) = NN_1(x) + C_1 e^{NN_2(-\frac{x-\alpha}{\varepsilon})} + C_2 e^{NN_3(\frac{x-\beta}{\varepsilon})} \quad (8)$$

We define $\alpha = 0$ and $\beta = 1$ to be the boundaries of x. Then we are able to use the optimization method of PINN to train new models. See Eq.(9). We only need to optimize the parameters of the three networks simultaneously. The whole structure of the our TPINN model is shown in Fig.2.

$$\begin{aligned} L(x;\theta_{new}) = &MSE\left(\mathcal{D}\left(NN_{new}(x)\right) - f(x)\right) \\ &+ MSE\left(NN_{new}(x) - B\right), \quad x \in [\alpha, \beta]. \end{aligned} \quad (9)$$

4 NUMERICAL EXPERIMENTS

4.1 Ordinary differential equations

A linear singularly perturbed differential equation without turning point is given as follows.

$$\begin{aligned} -\varepsilon u'' + xu' &= e^x \quad x \in (0,1), \\ u(0) = u(1) &= 0. \end{aligned} \quad (10)$$

On the right of u is the boundary layer. Here we set the number of PINN model layers to 3 and the number of nodes to (64, 64, 1). For TPINN, NN_1 is set to the same structure as PINN. NN_2 and NN_3 use smaller networks with the same structure as (16, 1). We choose Adam optimizer with a learning rate of 0.001. We use the hypercube to sample 1,000 data in $(0, 1)$, and fix the u of the two data points on the boundary to 0. We plan to train 10,000 epochs and observed the convergence of the two models. And we test another random 10000 data in $[0, 1]$ on both two models. In this equation we set ε to $\frac{1}{2^6}$. Objective function see Eq.(9). While two methods trained completely, the predicts of two models is shown in Fig.3. It can be seen from the Fig.3 that our method can improve the accuracy at the boundary layer. PINN fails completely when ε is very small. In other hand, our method converges faster because we have specified the approximate optimization direction in advance. PINN cannot learn anything during the 10,000 epochs and can be considered as a blind search.

Then we change the example to one that includes a turning point $x = 0$

$$\begin{aligned} -\varepsilon u'' + xu' &= \sin(\pi x) \quad x \in (-1,1), \\ u(-1) = u(1) &= 0. \end{aligned} \quad (11)$$

In this example, the solution u contains two boundary layers both on the left and right side. When the coefficient at the first derivative is negative in Eq.(11), the solution u contains an inner layer at the turning point $x = 0$, which we do not consider here.[2] We intend to set ε to $\frac{1}{2^4}$ and do not make it too small, because when we set ε to less than $\frac{1}{2^4}$, the boundary layer of the solution u will be too tortuous and not suitable for optimization, see Fig.4.

Here we set the model configuration as above. The only difference is that the value range of x is changed to $[-1, 1]$. Since the equation becomes more difficult, we increase the epoch to 20,000. As can be seen from the Fig.4, our method can still improve the prediction accuracy at the two boundary layers. TPINN converged more faster than traditional PINN because we have given a prior for the solution at the beginning.

4.2 Partial differential equations

We consider a elliptic differential equations with boundary problem,

$$\begin{aligned} -\varepsilon\Delta u + u_x + u_y &= 2\pi^2 \sin(\pi x)\sin(\pi y) \quad x \in (0,1) \text{ and } y \in (0,1), \\ u(0,y) = u(1,y) &= 0, \\ u(x,0) = u(x,1) &= 0. \end{aligned} \quad (12)$$

We set ε to 0.01. In this example, the boundary layer will appear at $x = 1$ and $y = 1$. So we set up 4 additional small networks to fit the layers on the four boundaries. The correct solution for this example is shown in Fig.5. Here we Use the hypercube to sample 1,000 coordinate points in $(0, 1)^2$, and randomly sample 100 points at each boundary, that is, a total of 400 boundary points and 1,000 interior points. The configuration of the two models remains unchanged, as in the first subsection 4.1. We plan to train 20,000 epochs and observe its convergence and predictions. See in Fig.6. Obviously, our method is more effective, and only 10,000 epochs can achieve the desired outcome.

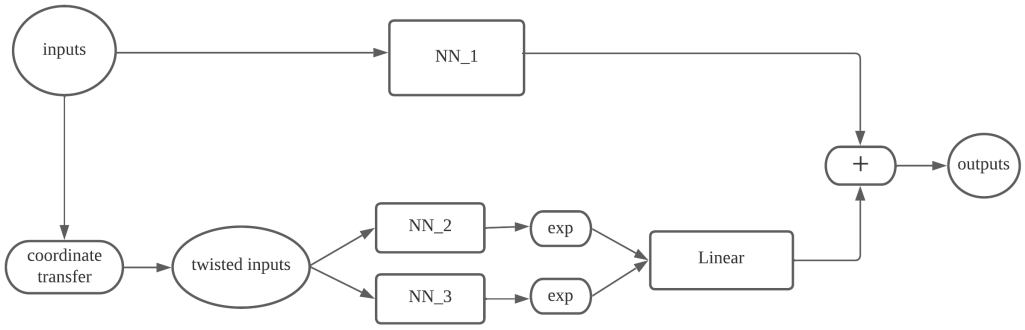

Figure 2: Overall structure and feedforward logic of our method TPINN.

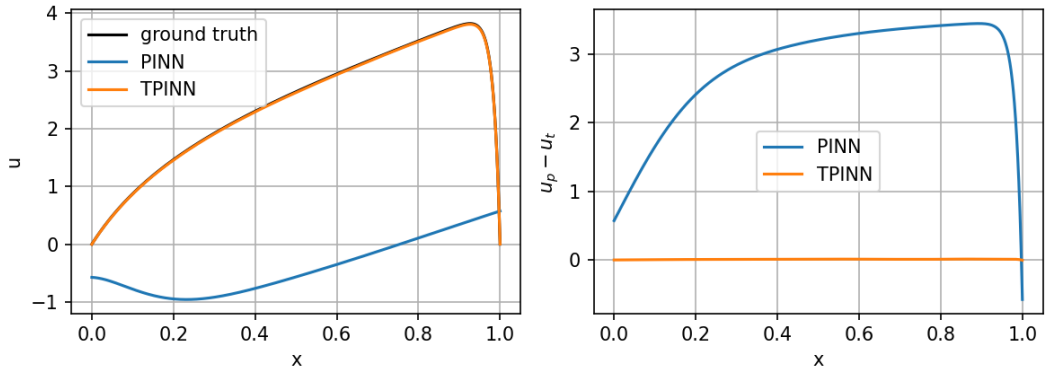

Figure 3: The predicts and residuals of both two methods on Eq.(10). u_p: the predicts of one model. u_t: the actual solution of the equation.

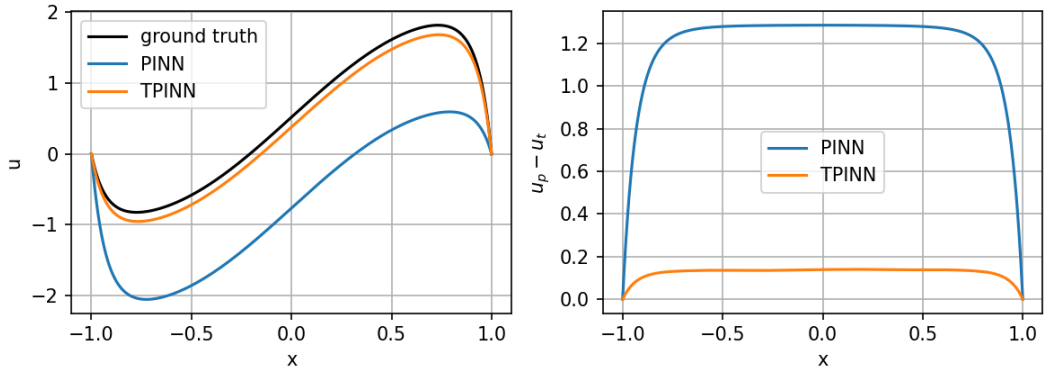

Figure 4: The predicts and residuals of both two methods on Eq.(11). u_p: the predicts of one model. u_t: the actual solution of the equation.

5 DISCUSSION

Although TPINN can greatly help solve the boundary layer problem, it still relies on PINN. It can't get rid of a series of other shortcomings of PINN, such as its performance is partly dependent on internal selection(the selection of the training data). So we can see that the effect of TPINN is not perfect.

In fact, differential equations are diverse, and any subtle changes to the equation may affect the neural network. A neural network

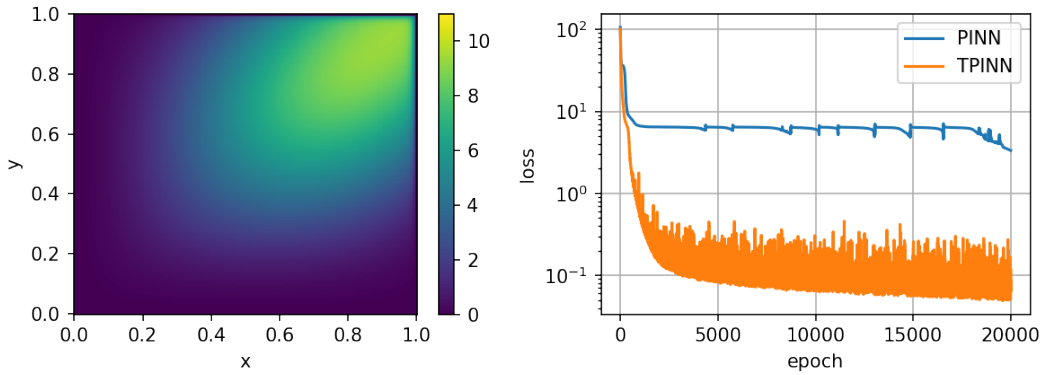

Figure 5: Left: The ground truth solution of Eq.(12). Right: The training loss of both two methods on Eq.(12).

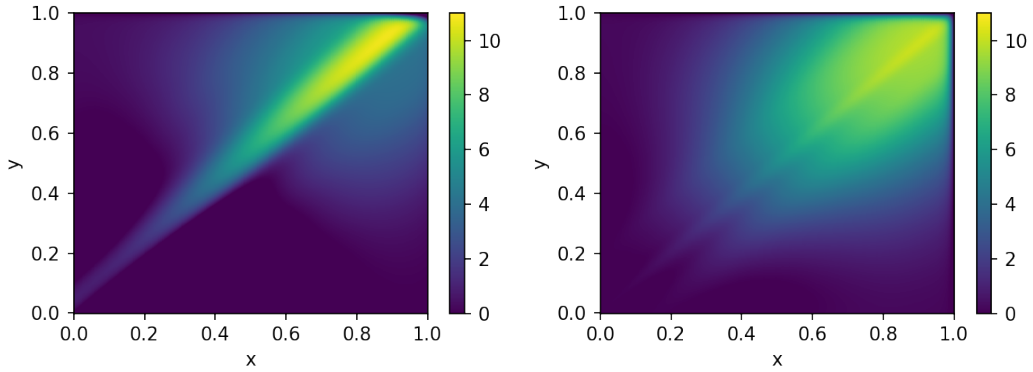

Figure 6: The predicts of both two methods on Eq.(12) after training 20000 epochs. Left: PINN, Right: TPINN.

is a point estimation tool and its performance is affected by many factors. For example, the selection of matching points, the setting of network structure, the method of updating network parameters, and the setting of the objective function. These problems are inescapable when we use neural networks to solve differential equations. But it is undeniable that neural networks will be faster than traditional numerical solutions on more complex equations.

The method of adding more prior knowledge starts from the equation. When comes to a nonlinear differential equation, we are impossible to know the actual structure of its solution. But we can use linear knowledge as a reference. There is no universal prior term that can be applied to all differential equations. For example, the layer can not only be exponential, but also can be parabolic. Otherwise, layer will not only appear at the boundary, but may also appear in the interior. This paper only studied some examples of exponential boundary layers. The reason why our method is feasible on examples of this paper is that we know the prior information. So the technology of adding more general prior information will be a challenging task in the future.

6 CONCLUSION

In this paper, we studied why PINN fails for some differential equations, mainly for singular perturbation differential equations, and proposed a TPINN method to overcome it. We found some failures are originated from the saddle point or local minimum point. By adding some prior terms to the solution may solve the problem. Through the study of the structure of the boundary layer, we add useful information of the boundary layer into the neural network. Experiments show that our TPINN method greatly improves the neural networks in solving singularly perturbed equations.

ACKNOWLEDGMENTS

This work was supported by NSFC project No.62106103, Fundamental Research Funds for the Central Universities No.ILA22023, Start-up Fund of Nanjing University of Aeronautics and Astronautics No.90YAH20131, 173 Program Technical Field Fund No.2021-JCJQ-JJ-0018.

REFERENCES

[1] Renardy M, Rogers R C (2006) An introduction to partial differential equations. Springer Science & Business Media.
[2] Roos H G, Stynes M, Tobiska L (2008) Robust numerical methods for singularly perturbed differential equations: convection-diffusion-reaction and flow problems. Springer Science & Business Media.
[3] Holmes M H (2012) Introduction to perturbation methods. Springer Science & Business Media.
[4] Duffy D J (2013) Finite Difference methods in financial engineering: a Partial Differential Equation approach. John Wiley & Sons.

[5] Mbroh N A, Munyakazi J B (2019) A fitted operator finite difference method of lines for singularly perturbed parabolic convection–diffusion problems. Math Comput Simulat 165: 156-171.
[6] Farhloul M, Mounim A S (2005) A mixed-hybrid finite element method for convection–diffusion problems. Appl Math Comput 171(2): 1037-1047.
[7] Li Y (2016) An Adaptive Finite Element Method with Hybrid Basis for Singularly Perturbed Nonlinear Eigenvalue Problems. Commun Comput Phys 19(2): 442-472.
[8] Lagaris I E, Likas A, Fotiadis D I (1998) Artificial neural networks for solving ordinary and partial differential equations[J]. IEEE Transactions on Neural Networks 9(5): 987-1000.
[9] Jafarian A, Mokhtarpour M, Baleanu D (2017) Artificial neural network approach for a class of fractional ordinary differential equation. Neural Comput Appl 28(4): 765-773.
[10] Rizaner F B, Rizaner A (2018) Approximate solutions of initial value problems for ordinary differential equations using radial basis function networks. Neural Process Lett 48(2): 1063-1071.
[11] Ramuhalli P, Udpa L, Udpa S S (2005) Finite-element neural networks for solving differential equations. IEEE Transactions on Neural Networks 16(6): 1381-1392.
[12] Liu H, Xing B, Wang Z, et al (2020) Legendre neural network method for several classes of singularly perturbed differential equations based on mapping and piecewise optimization technology. Neural Process Lett 51(3): 2891-2913.
[13] Raissi M, Perdikaris P, Karniadakis G E (2019) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378: 686-707.
[14] Raissi M, Yazdani A, Karniadakis G E (2020) Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science 367(6481): 1026-1030.
[15] Jin X, Cai S, Li H, et al (2021) NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. J Comput Phys 426: 109951.
[16] Mao Z, Jagtap A D, Karniadakis G E (2020) Physics-informed neural networks for high-speed flows. Comput Method Appl M 360: 112789.
[17] Sahli Costabal F, Yang Y, Perdikaris P, et al (2020) Physics-informed neural networks for cardiac activation mapping. Frontiers in Physics 8: 42.
[18] Fang Z, Zhan J (2019) Deep physical informed neural networks for metamaterial design. IEEE Access 8: 24506-24513.
[19] Chen Y, Lu L, Karniadakis G E, et al (2020) Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Opt Express 28(8): 11618-11633.
[20] Lu L, Meng X, Mao Z, et al (2021) DeepXDE: A deep learning library for solving differential equations. SIAM Rev 63(1): 208-228.
[21] Hennigh O, Narasimhan S, Nabian M A, et al (2020) NVIDIA SimNet^{TM}: An AI-accelerated multi-physics simulation framework. arXiv preprint arXiv:2012.07938.
[22] Cai S, Wang Z, Wang S, et al (2021) Physics-informed neural networks for heat transfer problems. J Heat Transfer 143(6): 060801.
[23] Wessels H, Weißenfels C, Wriggers P (2020) The neural particle method–an updated Lagrangian physics informed neural network for computational fluid dynamics. Comput Method Appl M 368: 113127.
[24] Meng X, Li Z, Zhang D, et al (2020) PPINN: Parareal physics-informed neural network for time-dependent PDEs. Comput Method Appl M 370: 113250.
[25] Yang L, Meng X, Karniadakis G E (2021) B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. J Comput Phys 425: 109913.
[26] Fang Z (2021) A high-efficient hybrid physics-informed neural networks based on convolutional neural network. IEEE T Neur Net Lear. https://doi.org/10.1109/TNNLS.2021.3070878
[27] Kharazmi E, Zhang Z, Karniadakis G E (2021) hp-VPINNs: Variational physics-informed neural networks with domain decomposition. Comput Method Appl M 374: 113547.