Proceedings of CONF-CIAP 2026 Symposium: International Conference on Atomic Magnetometer and Applications
DOI: 10.54254/2753-8818/2026.HZ28752

Physics-Informed Neural Network to Solve Singular Perturbation Problem

Shiyao Gu[1*], Jierui Li[2]

[1]*Mathematical Institute, University of Oxford, Oxford, UK*
[2]*Department of Physics, University of Cambridge, Cambridge, UK*
**Corresponding Author. Email: shiyao_liverpool@163.com*

Abstract. Singular perturbation problems are widely used in the field of fluid dynamics, but they pose a continuous challenge to conventional numerical methods because their solutions exhibit multiscale properties within the boundary layer. Although physically informed neural networks (PINNs) show great potential in solving partial differential equations, this paper proposes a novel PINN framework, which centers on the introduction of a learnable monotonic coordinate transformation that adaptively resolves the boundary-layer scales; a solution decomposition strategy that explicitly constructs the solution as the sum of the external and internal solutions and strictly satisfies the boundary conditions by means of a hard-coded approach; in addition, a novel loss function is designed to partition the weighted training mechanism to balance the loss magnitude between internal and external regions. In this Theoretical analysis reveals the amplification effect of the residuals in the boundary layer region, and the experimental results cover a variety of combinations of boundary conditions. Numerical experiments show that, compared with traditional PINN, our model achieves a significant improvement in accuracy near the boundary layer and is robust to different types of boundary conditions. This study provides a reliable and efficient solution for applying deep learning to multi-scale problems with boundary layers.

Keywords: Physics-Informed Neural Network, Singular Perturbation Problem, Boundary Layer, Stretched Coordinate, Domain-Decomposed Loss Function

1. Introduction

Neural networks (NNs) in machine learning have made impressive achievements in recent years in areas such as computer vision, natural language processing, data-driven modeling and scientific computing. However, conventional neural networks are often purely data-driven and lack the ability to incorporate physical constraints, which limits their applicability in scientific computing where governing equations play a fundamental role. To address this limitation, Raissi, Perdikaris, and Karniadakis introduced the concept of Physics-Informed Neural Network (PINN) [1]. PINN is a framework for embedding physical law residuals (usually represented by partial differential equations) into loss functions. In this way, PINNs can enhance consistency with observational data and physical models. Since their introduction, PINNs have been successfully applied to a variety of

© 2025 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/2026.HZ28752

problems, including forward and inverse PDEs, fluid dynamics, heat transfer, solid mechanics, and control theory [2].

Singular perturbation problem (SPP) is a basic problem which contains a small parameter and this parameter cannot be approximately regarded as 0. Most often in applications, this parameter is denoted by ε . In SPP problem, rapid changes in the boundary layer are the most concern. Such problems naturally arise in a wide range of applications, including fluid dynamics (boundary-layer theory in aerodynamics), chemical kinetics (fast–slow reaction systems), optimal control, semiconductor modeling, and quantum mechanics [3].

PINN has difficulty capturing the boundary layer. Traditional PINN predicts solutions with severe deviations at the boundary layer locations, and the PINN predictions deviate from the exact solution. For example, when solving the one-dimensional convection-diffusion equation: $-\varepsilon u''(x)+u'(x)=0,\quad u(0)=0,\quad u(1)=1.$When$\varepsilon << 1$, The solution will have a boundary layer near $x=1$.

Secondly, the loss function becomes unbalanced at multiple scales: the residuals in the narrow boundary layer are easily masked by the error in the smooth outer region, making it difficult for the network to capture the dynamics at the boundary layer, resulting in a larger error in the loss function [4].

Today's research on SPP problems has the following main categories, at first, theoretical analysis problems, the solutions in SPP contain outer solutions and boundary layers, ordinary methods cannot give a globally consistent approximation, and the classical methods are to try asymptotic expansions, and composite expansions have been developed [3].

Secondly, for modern methods, it is to use machine learning with PINN, the main problem is that the traditional numerical method is difficult to be promoted in high dimensional/complex geometry, the solution is generally to add the boundary layer into PINN for consideration, coordinate transformation, adjusting the loss function, and combining with asymptotic approximation theory in order to enhance the ability of capturing the boundary layer [5].

2. Method

The standard form of the problem we aim to solve:

$$-\varepsilon u''(x)+b(x)u'(x)+c(x)u(x)=f(x) \tag{1}$$

This is a singular perturbation problem, where the boundary condition are $x\in(0,1), u(0)=\alpha\left(u^{'}(0)=\alpha\right), u(1)=\beta\left(u^{'}(1)=\beta\right)$. Specifically, the small parameter is multiplied in front of the highest-order derivative, which causes the order of the operator to degenerate as $\varepsilon\to 0$, thereby leading to the formation of a boundary layer [6].

2.1. Stretched coordinate (scaled variable)

Traditionally, to solve problems such as (1), one needs to introduce terms like $\frac{x}{\varepsilon}$ (corresponding to the left boundary layer) and $\frac{1-x}{\varepsilon}$ (corresponding to the right boundary layer). Here, we define the transformed coordinate as:

$$s(x)=\frac{\phi(x_{\text{ref}})-\phi(x)}{\varepsilon},\quad \phi'(x)>0 \tag{2}$$

DOI: 10.54254/2753-8818/2026.HZ28752

where x_{ref} represents the position of the boundary layer. Specifically, $x_{\text{ref}} = 0$ if the boundary layer is on the left side, and $x_{\text{ref}} = 1$ if it is on the right side. The function ϕ is a strictly increasing function that controls the transformation of the coordinate, and it needs to be learned by the network. Intuitively, this approach allows the network to automatically learn the thickness of the boundary layer and to adapt the coordinate transformation according to different types of PDEs.

2.2. Form of the solution

For problems of the form (1), the solution of the differential equation can be expressed as:

$$u_\varepsilon = u_{\text{out}} + u_{\text{in}} \tag{3}$$

where u_{in} denotes the inner solution, which captures the rapid variation of the solution in boundary-layer regions or singular regions, while u_{out} represents the outer solution, which describes the smooth behavior of the solution in the regular domain away from singularities. This decomposition is a classical idea in singular perturbation theory and matched asymptotic expansions [7]. Our neural network, in its final results, is capable of partially exhibiting the contributions of both u_{in} and u_{out} .

To be specific, we adopt a hard-coding strategy to guide the network in learning the forms of u_{in} and u_{out}[8]. Specifically, we consider four scenarios: On the left and right sides of the problem, the boundary conditions can be either Dirichlet boundary conditions or Neumann boundary conditions. From the perspective of code implementation, we actually consider eight different cases, since the boundary layer can appear either on the left side or on the right side.

$$u_{\text{out}} = \begin{cases} \alpha + x g_{\text{out}}\ (x) & \text{Dirichlet} \\ \alpha x + \frac{x^2}{2}\, g_{\text{out}}\ (x) & \text{Neumann} \end{cases} \tag{4}$$

$$u_{\text{in}}\,(s) = \begin{cases} e^{-\lambda s}[(\beta - u_{\text{out}}\,(1)) + s g_{\text{in}}\,(s)] & \text{Dirichlet} \\ e^{-\lambda s}\left[(\beta - u'_{\text{out}}\,(1))s + \frac{s^2}{2}\, g_{in}(s)\right]\left(-\frac{\varepsilon}{\phi'(1)}\right) & \text{Neumann} \end{cases} \tag{5}$$

(This construction is not strictly necessary; we could directly use u_{out} and u_{in} for training. However, a factor of $e^{-\lambda s}$ must be added in front of u_{in} to ensure its rapid decay to zero from the boundary layer toward the non-boundary region. Otherwise, it would not be consistent with certain theoretical foundations used in the loss function section. In our implementation code, you can modify the parameter BC_MODE to choose either hard encoding, or the addition of penalty terms in the loss function to enforce that the predicted solution satisfies the boundary conditions.)

Here, in the construction of $u_{\text{out}}(x)$, we take the example where the boundary layer is on the right side. If the boundary layer is on the left side, it suffices to replace x with $1 - x$. Since in $u_{\text{in}}(s(x))$ the location of the boundary layer is embedded within $s(x)$, the form of u_{in} remains unchanged with respect to the position of the boundary layer. In our model, the network does not directly learn u_{out} and u_{in} , but instead learns $g_{\text{out}}(x)$ and $g_{\text{in}}(s)$, which ensures that the model can essentially satisfy the boundary conditions. The detailed derivation will be presented in the mathematical principles section later. In this setting, λ is also a parameter that needs to be learned during training. It is defined in the training process as: $\lambda = \text{softplus}(\theta) + \lambda_{\min}$[9]. The advantage of this design is that

DOI: 10.54254/2753-8818/2026.HZ28752

it not only constrains λ within a valid training range to keep it positive, but also provides smoother gradients.

2.3. Loss function

We first present the final loss function, and then we will explain the meaning of each component within it [1].

$$\mathscr{L} = E\left[w_{\text{in}}(s)(\varepsilon r_{\text{inner}})^2 + (1 - w_{\text{in}}(s))r_{\text{outer}}^2 + \lambda_{\text{comp}} w_{\text{band}}(s) r_{\text{comp}}^2\right] + \lambda_{\phi}\mathscr{L}_{\phi} + \lambda_{\text{BC}}\mathscr{L}_{\text{B}} \tag{6}$$

where $w_{\text{in}}(s) = \sigma(\kappa(S_0 - s))$, $w_{\text{band}}(s) = \sigma(\kappa(s - S_0))\sigma(\kappa(S_1 - s))$ (S_0, S_1 are hyperparameters).

We divide the solution domain, namely $[0,1]$, into three regions: the inner zone, the outer zone, and the transition band. We design two hyperparameters here: S_0 and S_1. Specifically, $0 < s < S_0$ corresponds to the inner zone, $s > S_0$ corresponds to the outer zone, and $S_0 < s < S_1$ corresponds to the transition band.

Inner zone: In the region close to the boundary layer, we only update $\theta_{\text{in}}, \theta_{\phi}, \theta_{\lambda}$, while keeping the outer solution frozen.

$$r_{\text{inner}}(x) = \mathscr{L}_{\varepsilon}[u_{\text{out}}(\text{ detach }) + u_{\text{in}}], \quad \tilde{r}_{\text{inner}} = \varepsilon r_{\text{inner}}, \quad \text{with weight } w_{\text{in}}(s) \tag{7}$$

where $\mathscr{L}_{\varepsilon}[u] := -\varepsilon u''(x) + b(x)u'(x) + c(x)u(x) - f(x)$. Moreover, in this region we adopt the scaled residual $\tilde{r}_{\text{inner}}$. This is done in order to keep the PDE losses from different regions on the same order of magnitude (which is also the main difficulty when using traditional Physics-Informed Neural Networks to solve such problems). We will later show that the order of r_{inner} is $O(1/\varepsilon)$.

Outer zone: In the region far away from the boundary layer, we only update $\theta_{\text{out}}, \theta_{\phi}, \theta_{\lambda}$, while keeping the inner solution frozen.

$$r_{\text{outer}}(x) = \mathscr{L}_{\varepsilon}[u_{\text{out}} + u_{\text{in}}(\text{detach})], \quad \text{with weight } (1 - w_{\text{in}}(s)) \tag{8}$$

Transition band: We treat the region near the boundary layer as the transition band, in which all parameters $\theta_{\text{in}}, \theta_{\text{out}}, \theta_{\phi}, \theta_{\lambda}$ are updated simultaneously.

$$r_{\text{comp}} = \mathscr{L}_{\varepsilon}[u_{\text{out}} + u_{\text{in}}], \quad \text{with weight } w_{\text{band}}(s) \tag{9}$$

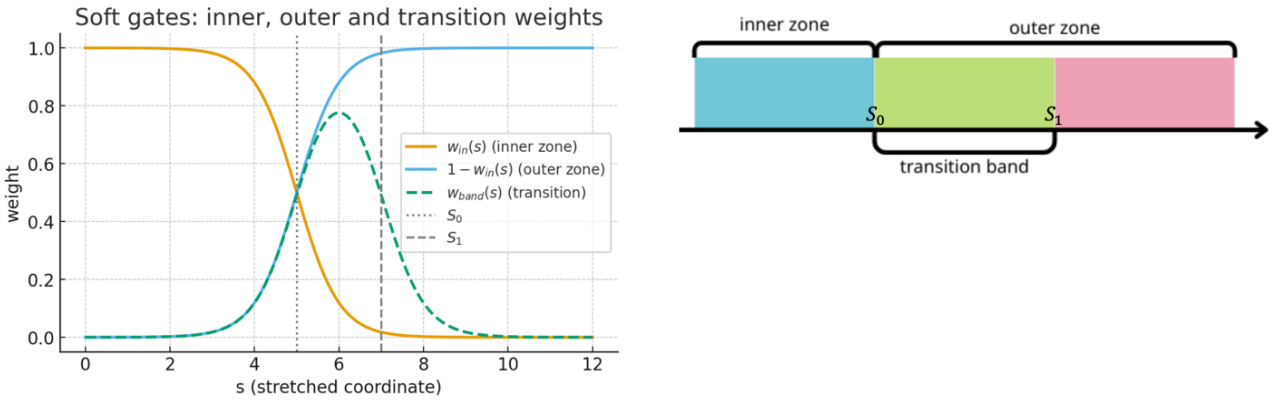
We aim for the following: in the inner zone, the training should mainly focus on the inner solution u_{in}, while in the outer zone, the training should mainly focus on the outer solution u_{out}. Therefore, we adopt a Sigmoid function to weight $\tilde{r}_{\text{inner}}$ and $\tilde{r}_{\text{outer}}$. In other words, the closer to the boundary layer, the more weight is assigned to $\tilde{r}_{\text{inner}}$, leading to more training of u_{in}, and conversely, more training of u_{out}. This design constitutes a “soft-gating” structure, whose advantage is that it makes the gradients of the loss function smoother, thereby facilitating subsequent training. As a simple example, suppose we adopt a “hard-gating” strategy. Then, in the interval $[0, S_0]$, the loss should be taken as $100\% r_{\text{inner}}$ and $0\% r_{\text{outer}}$, while outside this interval the opposite holds. However, under the “soft-gating” strategy, the closer we are to the boundary layer, the more weight is assigned to r_{inner}, and conversely, the less weight it receives when moving away from the boundary layer.

DOI: 10.54254/2753-8818/2026.HZ28752

Regarding the transition band: since in the training of the inner zone and the outer zone we respectively freeze the gradients of u_{out} and u_{in}, and adopt the Sigmoid function as the weighting mechanism, an issue arises. In the overlapping region, as shown in Figure 1, the weights of both $\tilde{r}_{\text{inner}}$ and $\tilde{r}_{\text{outer}}$ decrease simultaneously. This may result in the loss within this region being relatively larger compared to other regions. To compensate for this effect, we introduce r_{comp}.

The loss function of $\phi(x)$: To ensure that $s(x)$ remains monotonic and stable (without sudden changes), we impose a soft constraint on the controlling function $\phi(x)$[10]:

$$\mathcal{L}_{\phi} = \mathbb{E}\left[\max\left(0, \tau - \phi'(x)\right)^2\right] + \eta_{\phi}\mathbb{E}\left[\left(\phi''(x)\right)^2\right] \tag{10}$$



(a) Regional-based learning (b) Soft gate mechanism

Figure 1. Regional-based learning and soft gate mechanism

The loss function of the boundary conditions [1]:

$$\mathcal{L}_{\text{BC}} = \alpha_0(u(0) - \bar{u}_0)^2 + \alpha_1(u(1) - \bar{u}_1)^2 + \beta_0\left(u'(0) - \bar{q}_0\right)^2 + \beta_1\left(u'(1) - \bar{q}_1\right)^2 \tag{11}$$

The parameters $\alpha_0, \alpha_1, \beta_0, \beta_1$ depend on the type of boundary conditions: two of them are set to 0, and the other two are set to 1.

If hard-coding is not adopted, it is necessary to include $\mathcal{L}_{\text{BC}}$ in the loss function. In our implementation code, if BC_MODE = "soft" is set, the correct $\mathcal{L}_{\text{BC}}$ will be automatically incorporated into the loss function.

3. Results and discussion

3.1. Results

Firstly, here are introduce some symbols which are used in the code:

Table 1. Definition of symbols used in code

Symbols	Define
$NCOL$	Interior collocation points

DOI: 10.54254/2753-8818/2026.HZ28752

Table 1. (continued)

$NEDGE$	Boundary/edge-focused points
$EPOCHS_PER_STAGE$	Epochs per training stage (e.g., per- ϵ schedule)
$PRINT_EVERY, PLOT_EVERY$	Logging and visualization frequency (in steps)
LR	Optimizer learning rate
BC_MODE	How boundary conditions are imposed
LAM_BC	Boundary penalty weight used only when BC _ MODE = 'soft'
$S0, S1, kappa$	Region-splitting / gating parameters (inner / transition / outer)
lam_comp	Compatibility/consistency penalty of transition band
tau, eta_phi	Regularization for the learned coordinate transform $\phi(x)$
lam_pde, lam_phi	Weights for PDE residual and ϕ -regularization terms

These parameters are set in the Modified PINN code as:

Table 2. Value of parameters used in Modified PINN

Symbols	Value
$NCOL$	2048
$NEDGE$	512
$EPOCHS_PER_STAGE$	2000
$PRINT_EVERY, PLOT_EVERY$	100 , 500
LR	1 e -3
BC_MODE	'hard'
LAM_BC	1.0
$S0, S1, kappa$	5.0 , 7.0 , 2.0
lam_comp	0.05
tau, eta_phi	0.1 , 1 e -4
lam_pde, lam_phi	1.0 , 0.01

Specifically, we consider four cases. At the left and right boundaries of the SPP problem, the boundary conditions can be either Dirichlet boundary conditions or Neumann boundary conditions. However, in practice the exercise considered the fact that the boundary layer can appear both on the left and on the right, and we actually considered eight different cases. But here, just to show some examples of the more classical equations.

DOI: 10.54254/2753-8818/2026.HZ28752

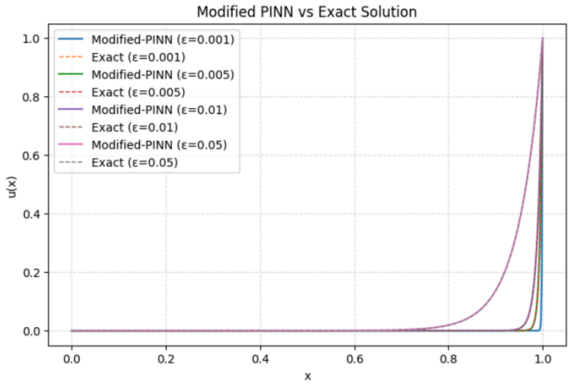
The display of Result is mainly shown by plotting the attempted solutions of the same equation by both the conventional PINN and the modified PINN.

Example 1: Both the left and right sides have Dirichlet boundary conditions, boundary layer on the right side.

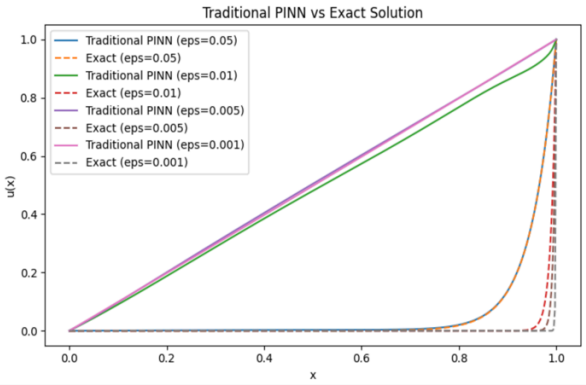
For example, when solving the one-dimensional convection-diffusion equation:

$$-\varepsilon u''(x) + u'(x) = 0. \quad u(0) = 0, \quad u(1) = 1 \tag{12}$$

When $0 < \varepsilon << 1$, The solution will have a boundary layer near $x = 1$.



(c) Traditional PINN vs exact solution



(d) Modified PINN vs exact solution

Figure 2. Traditional PINN and modified PINN vs exact solution for example 1

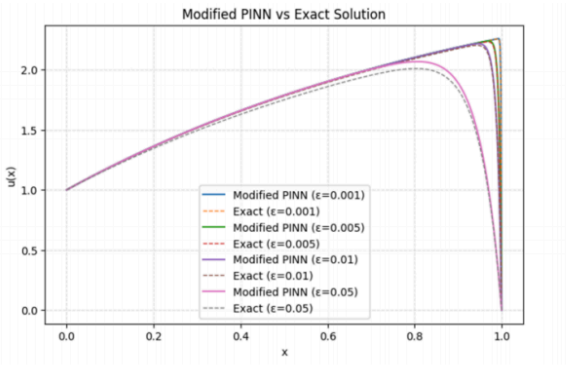
As shown in the Figure 2, we find that the Traditional PINN failed to approximate the solution near the sharp boundary $x = 1$.

Example 2: Both the left and right sides have Dirichlet boundary conditions, boundary layer on the right side.

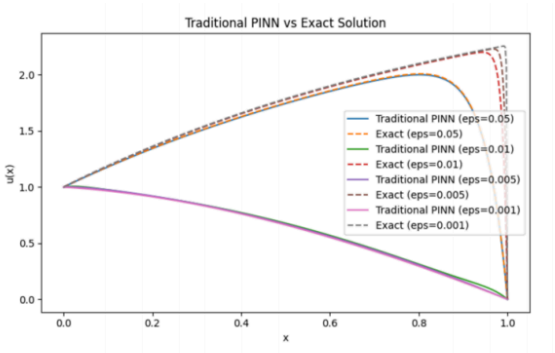
For example, when solving the one-dimensional reaction–diffusion equation:

$$-\varepsilon u''(x) + u'(x) = 2e^{-x}. \quad u(0) = 1, \quad u(1) = 0 \tag{13}$$

When $0 < \varepsilon << 1$, The solution will have a boundary layer near $x = 1$.



(e) Traditional PINN vs exact solution



(f) Modified PINN vs exact solution

Figure 3. Traditional PINN and modified PINN vs exact solution for example 2

DOI: 10.54254/2753-8818/2026.HZ28752

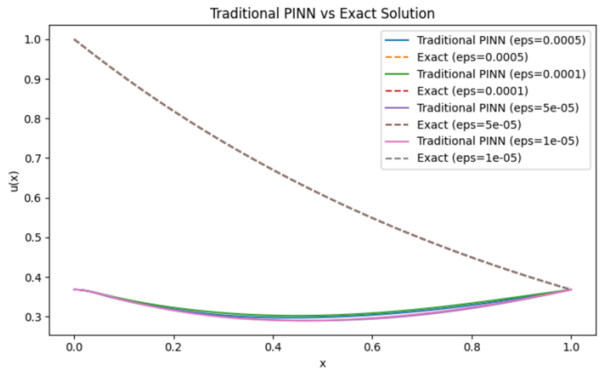
As shown in the Figure 3, we find that the Traditional PINN failed to approximate the solution near the sharp boundary x = 0.

Example 3: The left side has a Neumann boundary condition, and the right side has a Dirichlet boundary condition.

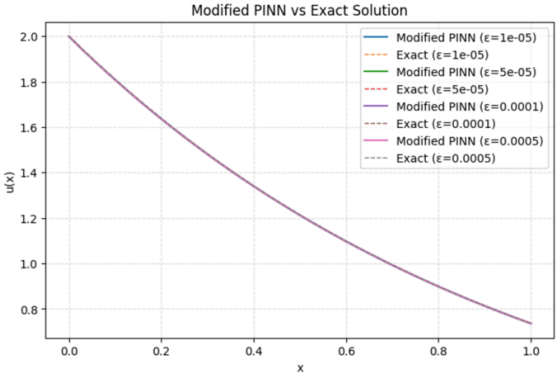
For example, when solving the one-dimensional convection-diffusion equation:

$$-\varepsilon u''(x) - u'(x) = 2e^{-x}. \quad u'(0) = 0, \quad u(1) = 2e^{-1} \tag{14}$$

When$0 < \varepsilon << 1$, The solution will have a boundary layer near$x = 1$.



(g) Traditional PINN vs exact solution



(h) Modified PINN vs exact solution

Figure 4. Traditional PINN and modified PINN vs exact solution for example 3

As shown in the Figure 4, we find that the Traditional PINN failed to approximate the solution near the sharp boundary x = 0.

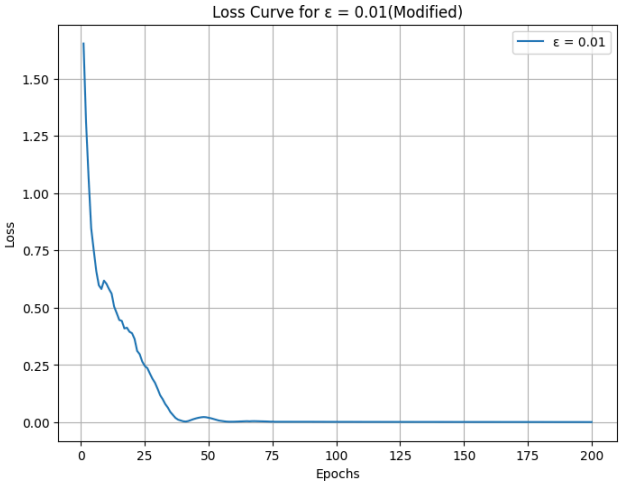
3.2. Discussion

The complete implementation code and additional examples can be found on our GitHub page: https://github.com/ShiyaoGu-2025/Physics-informed-Neural-Network-to-Solve-Singular-Perturbation-Problem

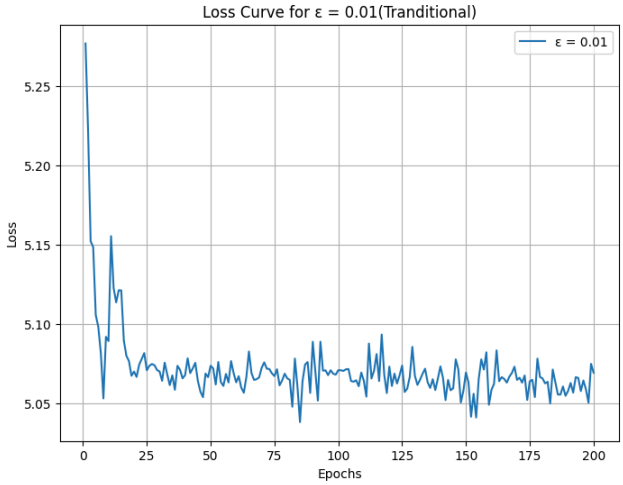
In our tests, we reached several conclusions and also identified some issues. First, as mentioned earlier, traditional PINN suffers from the gradient vanishing problem when solving such problems, while our improved PINN effectively addresses this issue. The following plot shows the training loss curves for both traditional PINN and the improved PINN:

This example is taken from Example 2 above, with $\varepsilon = 0.01$, and the first 200 training epochs are used. We can observe that the traditional method suffers from the gradient vanishing problem, which causes the loss function to stagnate, while our improved PINN loss function eventually converges almost to zero, as shown in Figure 5.

DOI: 10.54254/2753-8818/2026.HZ28752



(i) loss curve for tranditional PINN

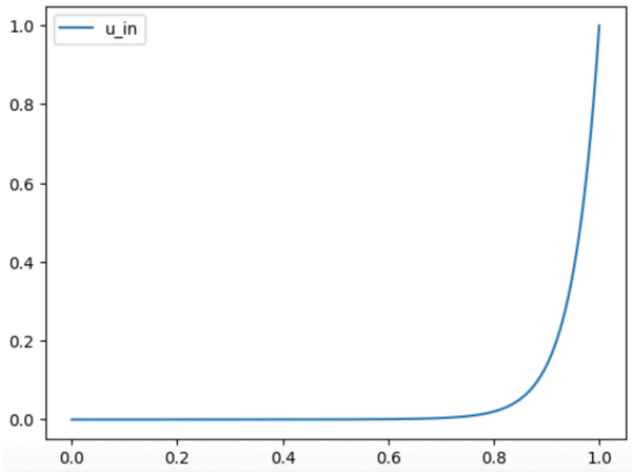


(j) loss curve for modified PINN

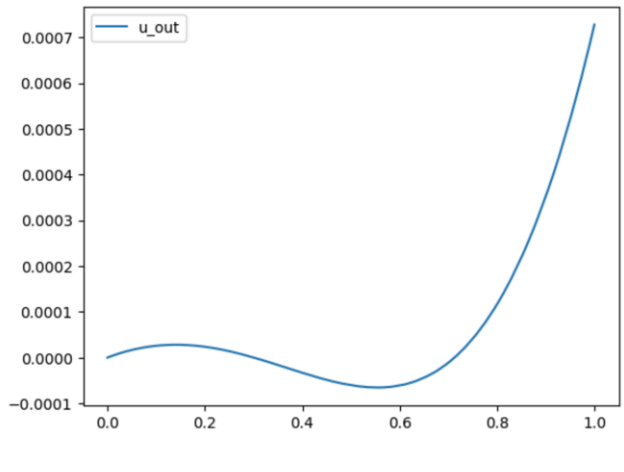
Figure 5. loss curve for tranditional PINN and modified PINN

Not all test cases exhibited this behavior, where the gradient descent of the loss function in the traditional method stalled. We also found some examples where the loss function of the traditional method converged relatively quickly. To date, no similar issues have been found with the improved method.

Furthermore, our method allows us to separately visualize u_{in} and u_{out}. For example, this is the result from Example 1(with $\varepsilon = 0.05$), where we can see that u_{out} is very close to the solution to the limit equation, which is $u = 0$, while u_{in} represents the boundary layer, with corrections applied on the right side, as shown in Figure 6.



(k) Predicted outer solution



(l) Predicted inner solution

Figure 6. Predicted outer solution and Predicted inner solution

To conclude, the core idea of the methods presented in the Method section can be summarized in one sentence: to help the neural network make an additional theoretical step. In other words, we inform the network about the properties that Singular Perturbation Problems should exhibit, thus narrowing the "search space" of the network, leading to smoother gradients, faster convergence, and

DOI: 10.54254/2753-8818/2026.HZ28752

other benefits. In fact, we have tried other seemingly reasonable strategies, but they did not yield effective results in practice. What we have retained are the strategies that showed better performance in one or more examples (e.g., boundary hard encoding, loss function partitioning, soft-gating strategy, etc.).

Regarding future directions of research, we believe that dividing the solution into several parts and theoretically studying the properties that each part should exhibit, along with region-wise solving, is likely an effective approach for using neural networks to solve many complex differential equations. For example, the approach presented in this paper could be extended to higher dimensions and more complex problems.

4. Mathematical principle

In this section, we provide the mathematical derivation of the principles underlying the method presented above.

(A): The constructed u_{in} and u_{out} essentially satisfy the boundary conditions, with only exponentially small errors on the side away from the boundary layer.

Now we make the following assumptions: (1) The boundary layer is located on the right side (the case for the left side is analogous). (2) After training, $s(x)$ becomes large (this is consistent with the solution behavior of problem (2.1), and our training results confirm this observation).

Case 1: Both the left and right sides have Dirichlet boundary conditions.

$$u(0) = u_{\text{out}}(0) + u_{\text{in}}(s(0)) = \alpha + \mathcal{O}(e^{-\lambda s(0)}), \quad u(1) = u_{\text{out}}(1) + u_{\text{in}}(0) = \beta \tag{15}$$

Case 2: The left side has a Dirichlet boundary condition, and the right side has a Neumann boundary condition.

$$\begin{aligned} u(0) = u_{\text{out}}(0) + u_{\text{in}}(s(0)) = \alpha + \mathcal{O}(e^{-\lambda s(0)}), \qquad & u^{'}(1) = u^{'}_{\text{out}}(1) + \tfrac{d}{ds} u_{\text{in}}(s)|_{s=0} \tfrac{ds}{dx} = \beta \\ \tfrac{ds}{dx} = -\tfrac{\phi^{'}(x)}{\varepsilon}, \quad \tfrac{d}{ds} u_{\text{in}}(s)|_{s=0} & = (\beta - u^{'}_{\text{out}}(1)) \cdot (-\tfrac{\varepsilon}{\phi^{'}_{(1)}}) \end{aligned} \tag{16}$$

Case 3: The left side has a Neumann boundary condition, and the right side has a Dirichlet boundary condition.

$$\begin{aligned} u'(0) = \alpha + \tfrac{d}{ds} u_{in}(s)|_{s=s(0)} \tfrac{ds}{dx} = \alpha + O(e^{-\lambda s(0)}), \; u(1) = u_{out}(1) + \quad & \beta - u_{out}(1) = \beta \\ \tfrac{ds}{dx} = -\tfrac{\phi'(x)}{\varepsilon}, \; \tfrac{d}{ds} u_{in}(s)|_{s=s(0)} = e^{-\lambda s}[\ldots]|_{s=s(0)} & = O(e^{-\lambda s(0)}) \end{aligned} \tag{17}$$

Case 4: Both the left and right sides have Neumann boundary conditions (in this case, the solution to problem (2.1) may not exist).

$$u^{'}(0) = \alpha + \frac{d}{ds} u_{in}(s)\Big|_{s=s(0)} \frac{ds}{dx} = \alpha + O\left(e^{-\lambda s(0)}\right), \quad u^{'}(1) = u^{'}_{out}(1) + \frac{d}{ds} u_{in}(s)\Big|_{s=0} \frac{ds}{dx} = \beta$$

$$\tfrac{ds}{dx} = -\tfrac{\phi'(x)}{\varepsilon}, \; \tfrac{d}{ds} u_{in}(s)|_{s=s(0)} = e^{-\lambda s}[\ldots]|_{s=s(0)} = O(e^{-\lambda s(0)}), \; \tfrac{d}{ds} u_{in}(s)|_{s=0} = (\beta - u'_{out}(1)) \cdot (-\tfrac{\varepsilon}{\phi'_{(1)}}) \tag{18}$$

(B): The order of r_{inner} is $O(1/\varepsilon)$.

DOI: 10.54254/2753-8818/2026.HZ28752

For convenience of notation, we denote $U(s(x)) := u_{\text{in}}(s(x))$, and let U_s and U_{ss} represent dU/ds and d^2U/ds^2 , respectively.

$$\mathscr{L}_\varepsilon[U] = -\varepsilon\left(\frac{\phi^{'2}}{\varepsilon^2}U_{ss} - \frac{\phi^{''}}{\varepsilon}U_s\right) + b\left(-\frac{\phi^{'}}{\varepsilon}U_s\right) + cU - f = -\varepsilon^{-1}\phi^{'2}U_{ss}$$

$$+\phi^{''}U_s - \frac{b\phi^{'}}{\varepsilon}U_s + cU - f = O\left(1/\varepsilon\right) \tag{19}$$

chain rule is used in the above equation:

$$\begin{aligned} &\frac{d}{dx} = \frac{ds}{dx}\frac{d}{ds}, \quad \frac{d^2}{dx^2} = \left(\frac{ds}{dx}\right)^2\frac{d^2}{ds^2} + \frac{d^2s}{dx^2}\frac{d}{ds} \quad \Rightarrow \quad \frac{ds}{dx} = -\frac{\phi^{'}(x)}{\varepsilon}, \quad \frac{d^2s}{dx^2} = -\frac{\phi^{''}(x)}{\varepsilon} \\ &\Rightarrow \quad U^{'}(x) = U_s\frac{ds}{dx} = -\frac{\phi^{'}}{\varepsilon}U_s, \qquad\qquad U^{''}(x) = \frac{\phi^{'2}}{\varepsilon^2}U_{ss} - \frac{\phi^{''}}{\varepsilon}U_s \end{aligned} \tag{20}$$

5. Conclusions

In this study, we introduced a novel Physics-Informed Neural Network (PINN) framework specifically designed to address singular perturbation problems exhibiting boundary layers. Central to our approach is a learnable, monotonic coordinate transformation that adaptively resolves fine-scale boundary-layer structures, coupled with a structured solution decomposition that inherently satisfies boundary conditions. Furthermore, we developed a region-wise loss weighting scheme that effectively balances the contribution of residuals from both inner and outer regions, mitigating training instability due to scale separation. Theoretic insights highlight the scaling behavior of residuals within boundary layers, providing a mathematical foundation for the proposed method. Extensive numerical experiments demonstrate that our model significantly outperforms conventional PINN in accuracy, particularly near boundary layers, and shows consistent robustness under diverse boundary condition types. This work establishes a reliable and efficient deep learning framework for solving multiscale boundary-layer problems, opening avenues for applying PINNs to more complex singularly perturbed models in fluid mechanics and beyond.

References

[1] Cuomo, Salvatore, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. 2022. “Scientific Machine Learning Through Physics–Informed Neural Networks: Where We Are and What’s Next.” Journal of Scientific Computing 92 (3): 88.

[2] Gupta, Maya, Andrew Cotter, Jan Pfeifer, Konstantin Voevodski, Kevin Canini, Alexander Mangylov, Wojciech Moczydlowski, and Alexander Van Esbroeck. 2016. “Monotonic Calibrated Interpolated Look-up Tables.” Journal of Machine Learning Research 17 (109): 1–47.

[3] Holmes, Mark H. 2012. Introduction to Perturbation Methods. Vol. 20. Springer Science & Business Media.

[4] Kevorkian, Jirair K, and Julian D Cole. 2012. Multiple Scale and Singular Perturbation Methods. Vol. 114. Springer Science & Business Media.

[5] Lagaris, Isaac E, Aristidis Likas, and Dimitrios I Fotiadis. 1998. “Artificial Neural Networks for Solving Ordinary and Partial Differential Equations.” IEEE Transactions on Neural Networks 9 (5): 987–1000.

[6] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. “Deep Learning.” Nature 521 (7553): 436–44.

[7] Nayfeh, Ali H. 2024. Perturbation Methods. John Wiley & Sons.

[8] Raissi, Maziar, Paris Perdikaris, and George E Karniadakis. 2019. “Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations.” Journal of Computational Physics 378: 686–707.

DOI: 10.54254/2753-8818/2026.HZ28752

[9] Wang, Long, Lei Zhang, and Guowei He. 2024. “Chien-Physics-Informed Neural Networks for Solving Singularly Perturbed Boundary-Layer Problems.” Applied Mathematics and Mechanics 45 (9): 1467–80.
[10] Wang, Sifan, Yujun Teng, and Paris Perdikaris. 2021. “Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks.” SIAM Journal on Scientific Computing 43 (5): A3055–81.