Component Fourier Neural Operator for Singularly Perturbed Differential Equations

Yiwen Pang, Ye Li*, Shengjun Huang
College of Computer Science and Technology/Artificial Intelligence,
Nanjing University of Aeronautics and Astronautics, Nanjing, China

August 2022

Abstract

Singularly perturbed differential equations (SPDEs) arise as mathematical models for various physical phenomena such as fluid flows and material sciences. The solutions of SPDEs usually contain special structure called layers. These layers are characterized by rapid transitions in the solutions, and are thus difficult to capture. With the development of machine learning, solving partial differential equations (PDEs) with neural network has been a new direction different from traditional numerical methods. This paper proposes a Component Fourier Neural Operator (ComFNO) to establish the mapping from any given source function of SPDEs to the solution of SPDEs. By incorporating asymptotic analysis knowledge into the design of neural network, our ComFNO improves the solution accuracy greatly. We also give a general structure for solving layer structures, applicable to other neural network-based PDE solvers. We test that our ComFNO has excellent adaptability to multiple data distributions. We also show that our ComFNO is suitable for the few-shot case, and the new structure can minimize the residual error of prediction even with very small amount of data.

Keywords: Neural operator; Singularly perturbed differential equations; Boundary layer

1 Introduction

Singularly perturbed differential equations (SPDEs) are a special class of differential equations in which there are small positive parameters ε in the highest derivative terms that make the solution of the equation change rapidly in a vary narrow region (usually at the boundary) [1]. This phenomenon is known as the famous boundary layer phenomenon proposed by Ludwig Prandtl in the last century. The research on singularly perturbed differential equations is very fragmented. The solution to SPDEs is rather hard to solve since it contains a very small parameter. Incorrect handling of small parameters can lead to large deviations in the results of the algorithm, whether traditional numerical methods or machine learning methods. For more information about SPDEs from ordinary differential equations, elliptic and parabolic problems, to the more realistic incompressible Navier-Stokes equations, please refer to the book [1].

Scientific machine learning (SciML) aims to use machine learning models to solve engineering problems such as differential equations. In recent years it is developed from physics-informed neural networks (PINNs) and other neural network-based PDE solvers [4, 3]. The PINN mainly designs an objective function for different differential equations and can use a small amount of data or even unsupervised training. It uses the physical information of the equation for training and lacks the

*Corresponding author. Email: yeli20@nuaa.edu.cn analysis of the real data distribution. In fact, the theory of scientific machine learning points to the need for using a large amount of physical information, but how much is not a definite concept. The development of scientific machine learning mainly solves the efficiency and generalization of traditional numerical methods such as finite difference, finite element, finite volume, etc. These methods are all discrete methods to a certain extent, and the accuracy mainly depends on the degree of dispersion. However, the denser the grid selection, the lower the algorithm efficiency. It is difficult to design different effective numerical methods for different practical problems. The emergence of deep learning shows that we do not necessarily need to extract data features independently. For new problems, deep learning can always explain the equation from the data level. Another drawback of PINNs is that it can only solve one specific PDE at a time. Once the initial/boundary conditions change, PINNs need to be retrained, thus are not effective for some engineering applications such as design and control.

In the last few years, there is a growing researches on neural operators [11], which can learn the infinite-dimensional mapping between functions and functions. In other words, we do not need to repeatedly solve different situations of an equation. This will lead to speed-up factors of thousands compared to vanilla PINNs and conventional numerical solvers. The Deep Operator Network (DeepONet) [10] and the Fourier Neural Operator (FNO) [9] have been proposed successively. DeepONet is mainly based on the universal approximation theorem of operators [23]. Given any function vector and the coordinate point, the value of the new function is mapped. DeepONet is mesh-invariant and does not specify a series of parameters such as classes of neural networks in the model, with theoretical convergence analysis [20, 24, 21] and applications [15, 16, 22]. However, it usually requires a very large and redundant data set to improve the prediction performance of the model. FNO is based on replacing the kernel integral operator by a convolution operator defined in Fourier space. FNO implements a series of the so-called Fourier layers computing global convolution operators with the fast Fourier transform (FFT) followed by mixing weights in the frequency domain and inverse Fourier transform. FNO is also guaranteed by theoretical error estimates [17] and the ability to simulate many physical phenomena such as fluid flows [19], seismic waves [25] and material modelling [18]. Although the performance of DeepONet and FNO is comparable for a number of important PDEs [15], we consider FNO in this paper due to its superior cost-accuracy trade-off [26].

Solving singularly perturbed differential equations is mostly in the research area of applied mathematics. But due to the widely use of scientific machine learning, the study of singularly perturbed differential equations is necessary. In [1], a series of variants of traditional numerical methods for solving singularly perturbed differential equations are listed, such as classical finite difference method and finite element method. They are mainly based on the progressive development of the solutions of different singularly perturbed differential equations, and are optimized, such as the adaptive mesh method of finite difference, to make the mesh of the boundary layer denser. These numerical methods require manual design of the grid in advance, and the solution accuracy is completely dependent on the density of the grid. When scientific machine learning became popular, some excellent research gradually emerged. [5] was the first to propose their basic principle of using neural networks to solve singularly perturbed differential equations. But it only works for specific 1D boundary problems, and for different equations, the objective function needs to be redesigned. On the basis of principle, an advanced method for solving SPDE by neural network is given [27]. It adds an adaptive grid method. When an area with too large residuals is found during training, the training points in this area will be doubled. This method is suitable for the one-dimensional case, and the adaptive grid is not suitable for ordinary operator networks, because the added data points are not supported by physical information. [6] proposed a Legendre neural network using mapping method and partition optimization method. A new mapping strategy is designed to automatically move the selected points closer to the boundary, but this method of selecting points is only applicable to the boundary layer situation. [7] proposed a SPDE-Net based on Galerkin method. This network predicts the SUPG(streamline upwind/Petrov-Galerkin) stabilization parameter, through that calculates the solution of the equation through the parameter. The net is optimized by the residuals of solution and the residuals of the parameter. Compared with PINN, the solution accuracy is improved, but it is still a numerical solution method in essence. [8] proposed an improved scheme of PINN. By transforming the original convection-diffusion equation, it is easier for PINN to solve the weak form equation. A variety of transformation equations are provided in [8]. Theoretically, the solution accuracy can be improved in the original PINN, but the actual performance is weak. This method does not apply to neural operators.

The solutions of SPDEs usually contains boundary layers, in which the solutions change rapidly, and are difficult to capture by vanilla FNO and other neural network-based PDE solvers.

This paper mainly improves FNO for solving SPDEs. Our contributions include:

- We propose a Component Fourier Neural Operator (ComFNO) for solving SPDEs. The asymptotic analysis knowledge of SPDEs is incorporated into the original FNO neural network architecture, so our ComFNO can capture boundary layers more accurately, and improves the solution accuracy greatly.
- We proposed a general framework for solving layer structures in SPDEs, applicable to other neural network-based PDE solvers.
- We experiment on different SPDEs for a range of the small parameter $\varepsilon > 0$, show that our ComFNO has excellent adaptability to multiple data distributions, and significantly reduce the mean and variance of the residual error of the predicted solutions.

2 Problem settings and Preliminaries

2.1 Singularly Perturbed Differential Equations

A simple convection-diffusion equation can be described in the following format [1]

$$\begin{gathered} -\varepsilon u'' + b(x)u' + c(x)u = f(x) \quad x \in (0,1)\,, \\ u(0) = u(1) = 0, \end{gathered} \tag{1}$$

where ε is a very small positive parameter. The higher-order term u'' is the diffusion term, and the first-order term u' is the convection term. Due to the existence of a small parameter ε, the solution of such an equation is generally badly behaved, as shown in Figure 1.

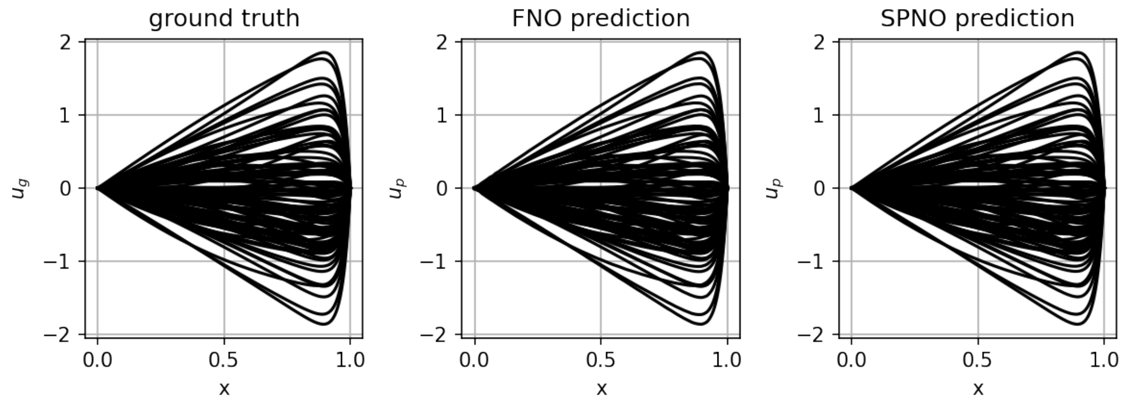


Figure 1: Ground truth and predictions for both FNO and our ComFNO. Here we have $b(x) = 1$ and $c(x) = 0$ in equation (1). We take 1000 different $f(x)$ to generate our training set and another 100 $f(x)$ for testing. We can see that, the true solutions of the equation (1) have a boundary layer at $x = 1$. The degree of distortion of boundary layer depends on the parameter ε and the function $f(x)$.

This class of problems with small parameters tending to zero are called singularly perturbed problems, where ε is the perturbation parameter. In the explanation of [1], the solution of the singularly perturbed differential equation contains a special structure called layer, which generally exists in a narrow region at the boundary. This type of region is called boundary layer. In the simple case given above, it can be seen that the boundary layer is at $x = 1$.

2.2 Fourier Neural Operator

The Fourier neural operator is derived from the kernel neural operator, which replaces the operator with the Fourier operator. We define F as the Fourier transform, F^{-1} is the inverse transformation.

$$\begin{aligned} (Ff)(k) &= \int_D f(x) e^{-2i\pi\langle x,k\rangle} dx, \\ \left(F^{-1}f\right)(x) &= \int_D f(k) e^{2i\pi\langle x,k\rangle} dx. \end{aligned} \tag{2}$$

[9] define the Fourier integral operator K

$$\left(K(\phi)v_t\right)(x) = F^{-1}\left(R_\phi \cdot (Fv_t)\right)(x), \tag{3}$$

where $R_\phi = F(\kappa_\phi)$. κ is a periodic function parameterized by ϕ. A Fourier integral operator is a basic unit of the Fourier neural operator.

2.3 Neural Network-based PDE Solver

According to [2], using machine learning methods to solve differential equations is classified as scientific machine learning. Current solvers can be roughly divided into two directions: PDE solvers and operator solvers. The PDE solver solve one specific PDE while the operator solver solve a class of equations without the need for retraining. The representative example of PDE solver is PINN and that of the operator solver is DeepONet and FNO. Karniadakis et al. [3] suppose many models are classified according to the amount of data used, and can be divided into three types. The first has no data-driven only physical information. The second is the opposite: only data-driven without physical information. The third is a compromise: possess both data and physical information. In fact, the most applied and most scientific should be the third one, because usually we only have a part of the data and only understand part of the physical knowledge in the system. PINN is a celebrated method that combines data and physical information to assist training. The difference between the operator solver and the PDE solver is mainly their generalization. The operator solver can directly solve a class of equations without the need for different model training based on different instances of a class of equations.

When we use PDE solvers to solve SPDEs, we argue that the useful information of physics-informed machine learning such as PINN is limited. They only add the information of known equations, which can be regarded as priors. The generalization of this method is not strong if it is aimed at a fine-grained type of equation. Extremely, when PINN is dealing with SPDEs, when ε is too small, PINN will fail. For other PINN failures could refer to [8, 12]. The main reason may be that there is no supporting data in the region.

3 Method

3.1 Asymptotic Expansions

[1] gives us a very specific proof of asymptotic expansion and here we only briefly state the relevant conclusions.

3.1.1 Ordinary Differential Equations

First let us consider a simple convection-diffusion equation [1]

$$\begin{aligned} -\varepsilon u'' + b(x)u' + c(x)u = f(x) \quad & \text{for } x \in (0,1)\,, \\ u(0) = & u(1) = 0, \\ c(x) \geq & 0, \text{ for } x \in [0,1]\,. \end{aligned} \tag{4}$$

When specify operator $L_0 v := bv' + cv$ when $\varepsilon = 0$, the equation can be divided into two parts: $L_0 u_0 = f$ and $L_0 u_\nu = u''_{\nu-1}$. The first part is called the reduced problem of this equation, and u_0 is called the reduced solution. If we want to dissect the solution of this singularly perturbed differential equation, we need to asymptotically develop the solution. For the above ordinary differential equation (without turning point), the expansion of the solution u_{as} is [1]

$$\begin{aligned} u_{as}(x) &= u_g(x) + v_{loc}(\xi) \\ &= \sum_{\nu=0}^{m} \varepsilon^\nu u_\nu(x) + \sum_{\mu=0}^{m} \varepsilon^\mu v_\mu(\frac{1-x}{\varepsilon}), \\ |u(x) - u_{as}(x)| &\leq C\varepsilon^{m+1} \quad \text{for } x \in [0,1] \text{ and } \varepsilon \leq \varepsilon_0. \end{aligned} \tag{5}$$

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.

[1] gives us an important lemma for solving singularly perturbed differential equations, specifying directions for our new structure. For the solution with boundary layer, we can split it into a smooth part S and a layer part E, then $u = S + E$. This decomposition is called S-type decomposition. For

instance in Eq.17, with $b(x) > \beta > 0$, define a positive integer q

$$\begin{aligned}
&|S^{(l)}(x)| \leq C, \\
&|E^{(l)}(x)| \leq C\varepsilon^{-l} \exp\left(-\frac{\beta(1-x)}{\varepsilon}\right) \quad \text{for } 0 \leq l \leq q, \\
&S\, satisfies\, LS = f, E\, satisfies\, LE = 0.
\end{aligned} \tag{6}$$

Things get complicated when there are turning points in the equation. When considering [1]

$$\begin{aligned}
-\varepsilon u'' + xb(x)u' + c(x)u = f(x) \quad x \in (-1, 1)\,, \\
u(-1) = u(1) = 0.
\end{aligned} \tag{7}$$

The turning point is at $x = 0$. We specify $b(x) \neq 0, c(x) \geq 0, c(0) > 0$, if $b(x) > 0$, then u will have two boundary layers, the boundary layer structure Similar to above (exponential function form). But when $b(x) < 0$, the boundary layer will not exist, we define $\lambda := -c(0)/b(0)$, then the reduced solution can be written as [1]

$$u_0(x) = \begin{cases} (|x|^k - |x|^\lambda)/(k - \lambda), & \lambda \neq k, \\ x^k \ln|x|, & \lambda = k. \end{cases} \tag{8}$$

There is an inner layer of u at $x = 0$, and when $\lambda = 0$, u will have an inner shock layer at $x = 0$, because u_0 will be discontinuous. So the lemma in this case can be given as [1]

$$|u^{(l)}(x)| \leq C(1 + |x| + \varepsilon^{1/2})^{\lambda - l}. \tag{9}$$

For higher-order equations, an asymptotic expansion can be given as [1]

$$\begin{aligned}
u_{as}(x) = \sum_{\nu=0}^{m} u_\nu(x)\varepsilon^\nu + \varepsilon^3 \left(\sum_{\mu=0}^{m} v_\mu(\xi)\varepsilon^\mu\right) e^{-x/\varepsilon} \\
+ \varepsilon \left(\sum_{\mu=0}^{m} w_\mu(\zeta)\varepsilon^\mu\right) e^{-(1-x)/\varepsilon}, \\
\text{for } \xi = x/\varepsilon,\ \zeta = (1 - x)/\varepsilon.
\end{aligned} \tag{10}$$

3.1.2 Partial Differential Equations

For the initial-boundary value problem of a parabolic partial differential equation in the space-time domain [1]

$$\begin{aligned}
u_t(x,t) - \varepsilon u_{xx}(x,t) + b(x,t)u_x(x,t) + d(x,t)u(x,t) = f(x,t), \\
(x,t) \in (0,1) \times (0,T], \\
u(x,0) = s(x) \quad 0 \leq x \leq 1, \\
u(0,t) = q_0(t) \quad 0 < t \leq T, \\
u(1,t) = q_1(t) \quad 0 < t \leq T,
\end{aligned} \tag{11}$$

Asymptotic expansion can be given as [1]

$$\begin{aligned}
u(x,t) = u_0(x,t) + v(x,t) + w(x,t), \\
\text{with } v(x,t) = -\tilde{u}_0(1,t)e^{-b(1,t)(1-x)/\varepsilon},
\end{aligned} \tag{12}$$

where u_0 is the reduced solution, v is the boundary layer function (exponential boundary layer), w is the bias term, $|w(x,t)| \leq C\varepsilon^{1/2}$.

For a boundary value problem of an elliptic partial differential equation in spatial domain [1]

$$\begin{aligned} -\varepsilon\Delta u + b(x,y)\cdot\nabla u + c(x,y)u &= f(x,y) \quad \text{in}\, \Omega = (0,1)\times(0,1), \\ u &= 0 \quad \text{on}\, \partial\Omega, \end{aligned} \tag{13}$$

the boundary layer is exponential, and the asymptotic expansion of u is [1]

$$\begin{aligned} u_{as}(x,y) &:= u_{as}^{*}(x,y) + u_0(1,1)\exp\left[-b_1(1,1)\frac{1-x}{\varepsilon}\right]\exp\left[-b_2(1,1)\frac{1-y}{\varepsilon}\right], \\ u_{as}^{*}(x,y) &:= u_0(x,y) - u_0(1,y)\exp\left[-b_1(1,y)\frac{1-x}{\varepsilon}\right] \\ &\quad - u_0(x,1)\exp\left[-b_2(x,1)\frac{1-y}{\varepsilon}\right]. \end{aligned} \tag{14}$$

It can be seen that there are two exponential boundary layers at $x = 1$ and $y = 1$. Because the two boundary layers are superimposed at $(1,1)$, it is necessary to add a corner layer correction. In the partial differential equation, there may be a situation of $b \equiv 0$, a parabolic boundary layer, also called a characteristic boundary layer, exists at both $x = 0$ and $x = 1$. For example in this case [1]

$$\begin{aligned} -\varepsilon\Delta u + \frac{\partial u}{\partial y} &= f \quad \text{in}\, \Omega = (0,1)\times(0,1), \\ u &= 0 \quad \text{on}\, \partial\Omega. \end{aligned} \tag{15}$$

It is more complicated to study the parabolic boundary layer, but fortunately [1] came to a conclusion: temporarily ignoring the parabolic boundary layer and only considering the exponential boundary layer can make the deviation reach $O(\varepsilon^{1/4})$, its gradually develops into [1]

$$u_{as}(x,y) := u_0(x,y) - u_0(x,1)\exp\left(-\frac{1-y}{\varepsilon}\right). \tag{16}$$

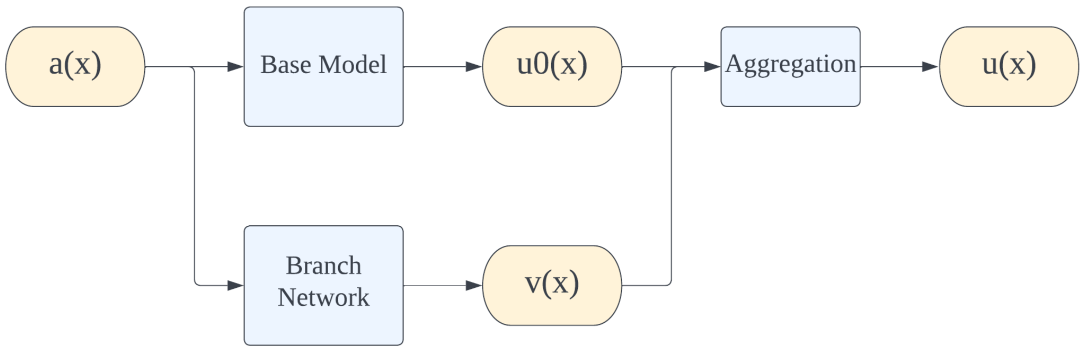


Figure 2: $a(x)$ and $u(x)$ represent the input of model and solution of the problem. The upper part of the model is a traditional neural operator. The branch network mainly fit the boundary layer function which will assist neural operator.

3.2 Improved Network Structure

According to the progressive development of the solution in the previous section, a general framework can be independently designed for the boundary layer. See Fig.2. The main work of the branch network is residual completion for basic model.

Similarly, if we use the FNO or other neural operators, it can be designed in the following format. See Fig.3. This is only for the simplest exponential boundary layer, because the exponential boundary layer occurs most frequently and has the greatest impact on the solution. In fact this model is free to add any version of the boundary layer. However, it is enough to give the approximate structure of the exponential boundary layer for neural network. The remaining parts of the solution will be fitted by the rest of the neural network modules. Generally speaking, these parts are easy to fit (smooth part). In next section, some experiments have been carried out to prove that the new structure has obvious advantages in improving the overall accuracy.

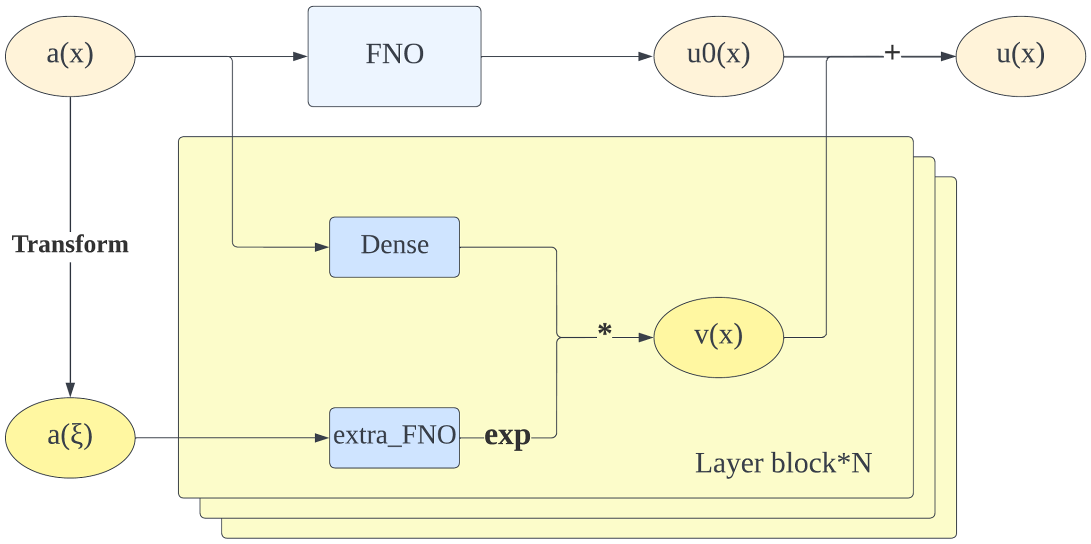


Figure 3: $a(x)$ and $u(x)$ represent the input of model and solution of the problem. The upper part of the model is traditional Fourier neural operator. $a(\xi)$ means we need to transform the coordinate to the boundary. For example we can transform x to the boundary $x = 1$ by using $\xi = \frac{1-x}{\varepsilon}, for\, x \in (0,1)$. The layer blocks mainly fit the boundary layer functions which will assist Fourier neural operator. The recommended number of layer blocks is 2 for 1-dimension, 4 for 2-dimensions.

4 Experiments

4.1 Settings

In this subsection, the configuration of the experiment will be given. The baseline we use is a FNO with 4 Fourier layers[9]. In one-dimension situation width of network is 64 and of filter is 16, and in two-dimension width of network is 16 and of filter is 12. At the data level, we use the Gaussian random fields[10, 9, 13]. The kernel function is radial basis function kernel which length scale fits, to generate 1000 random samples which resolution is 1001. These 1000 samples are used to solve different differential equations by the finite difference method to obtain 1000 solutions. We select batch size = 50, epoch = 500 in 1-dimensional case, batch size = 20, epoch = 800 in 2-dimensional case. All experiments use Adam optimizer, learning rate = 0.001. All objective functions choose l2 relative error. Most of these configurations are derived from [9].

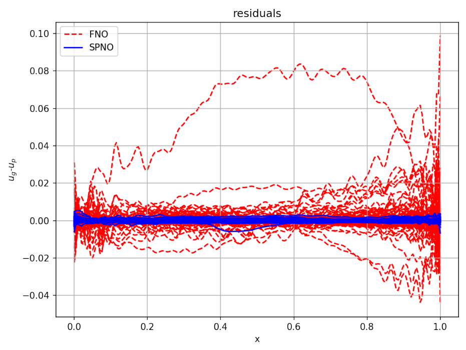
The control group uses our method, where the trunk network is the same as the baseline and the branch network uses exponential boundary layer blocks by default. In one-dimensional, the number of layer blocks is 2. Taking $x \in (0,1)$ as an example, $\xi_1 = T_1(x) = (1-x)/\varepsilon$, $\xi_2 = T_2(x) = -x/\varepsilon$. In two-dimensional, the number of layer blocks is still 2. Taking $\boldsymbol{x} \in (0,1)^2$ as an example, $\xi_1 = T_1(x) = (1-\boldsymbol{x})/\varepsilon$, $\xi_2 = T_2(x) = -\boldsymbol{x}/\varepsilon$.

4.2 Ordinary Differential Equations

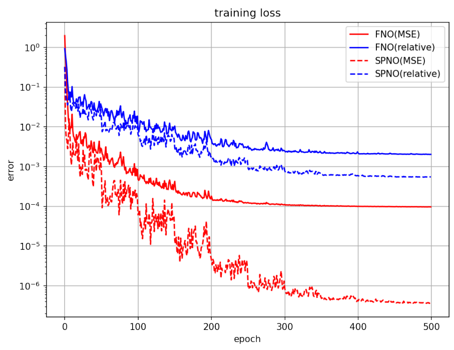
For ordinary differential equations, we mainly tested the example without turning point. For the case that with turning point, we found an example that contains the boundary layer and performed a simple test. Consider

$$\begin{aligned} -\varepsilon u'' + xu' &= f \quad x \in (0,1), \\ u(0) = u(1) &= 0. \end{aligned} \tag{17}$$

This is the simplest ordinary differential equation, and we wanted to see if our added boundary layer structure would perform particularly well in simple cases. The boundary layer of this example is the simplest exponential boundary layer, containing only a decaying exponential function at $x = 1$. We want to see if FNO will perform well in this example, because FNO works well for most equations which task has a lot of data support. In [7], the authors pointed out that the FNO model is resolution-insensitive, and even lower resolutions can achieve better results. However, as shown in the Fig.4(a), the residual of FNO in the boundary layer is larger than the rest of the region. Because when ε tends to 0, the amount of data near the boundary is very small, and the change is very large, the model will be under-fitting for the data of the boundary layer.



(a) residuals on both two methods



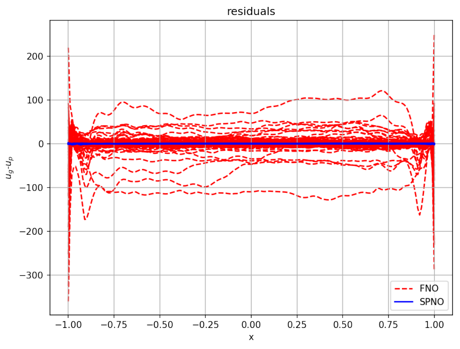
(b) training loss for both two methods

Figure 4: Performance of both FNO and SPNO on Eq.17. (a) The residuals is calculated by $u_g - u_p$. u_g is the ground truth and u_p is the predicts of the model. (b) The relative loss and MSE are both presented.

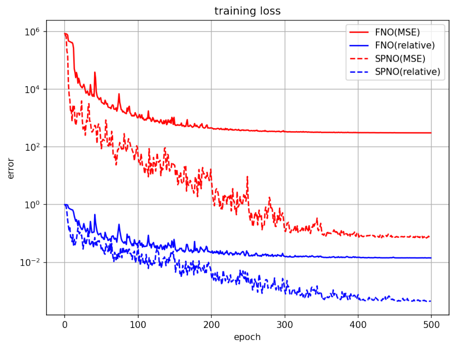
In fact, the effect of FNO has achieved relatively high accuracy in most area in this simple example, approximately 10^-2 MAE. But in order to verify our method, we will still use several relatively simple examples to ensure the generalization of our model. We can see from the residuals of FNO certain periodic fluctuations occur. After adding the layer blocks, the residuals for the stable region also tend to decrease. For this simple case, it can be shown that our layer blocks structure conforms to the structure of the solution to this differential equation. In other words, this structure can improve not only the layer phenomenon at the boundary, but also reduce the overall error.

Then we consider the case with turning point

$$\begin{aligned} -\varepsilon u'' + xu' &= f \quad x \in (-1,1), \\ u(-1) = u(1) &= 0. \end{aligned} \tag{18}$$



(a) residuals on both two methods



(b) training loss for both two methods

Figure 5: Performance of both FNO and SPNO on Eq.18. (a) The residuals is calculated by $u_g - u_p$. u_g is the ground truth and u_p is the predicts of the model. (b) The relative loss and MSE are both presented.

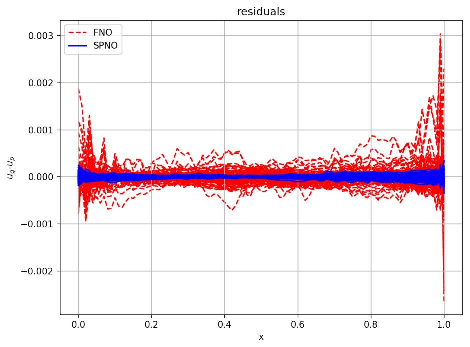
There are 2 boundary layers for this equation, which exist on both boundaries. In this situation, the two layer blocks we added will all play a role, as shown in the Fig.5(a). The boundary layer of this equation is usually more distorted, and the residual of FNO at the boundary layer will be greater than 1. In this experiment we set $\varepsilon = \frac{1}{2^4}$, where ε too small will cause the gradient of the solution to explode.

4.3 Partial Differential Equations

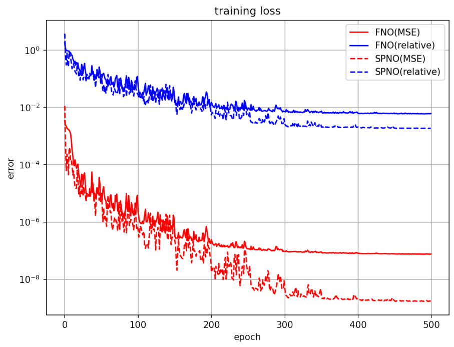
Here we consider a parabolic differential equation with initial-boundary problem in space-time domain,

$$\begin{aligned} u_t - \varepsilon u_{xx} + u_x &= 0 \quad x \in (0,1)\,\text{and}\,t \in (0,T], \\ u(x,0) &= f(x), \\ u(0,t) &= u(1,t) = 0. \end{aligned} \tag{19}$$

We want to get the mapping of $u(x,0) \to u(x,1)$ so this is still a one-dimensional example. As time t increases, $u(x,t)$ will gradually generate a boundary layer. The boundary layer is the largest until $t = T$. In Fig.6(a), we can see that the residuals are very similar to the ODE example. Even from ODE to PDE, the nature of the mapping of the solution does not change.



(a) residuals on both two methods



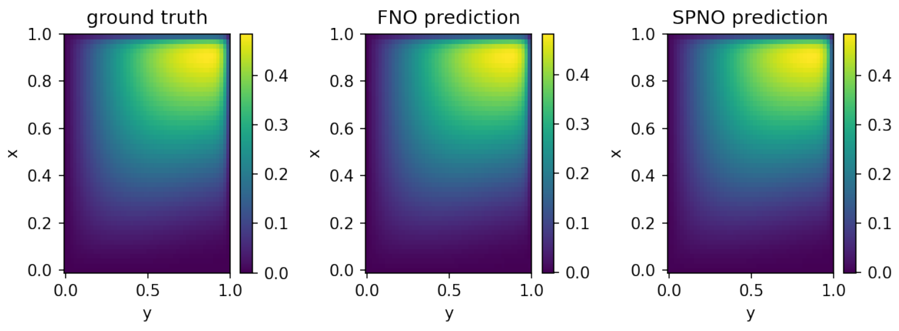
(b) training loss for both two methods

Figure 6: Performance of both FNO and SPNO on Eq.19. (a) The residuals is calculated by $u_g - u_p$. u_g is the ground truth and u_p is the predicts of the model. (b) The relative loss and MSE are both presented.

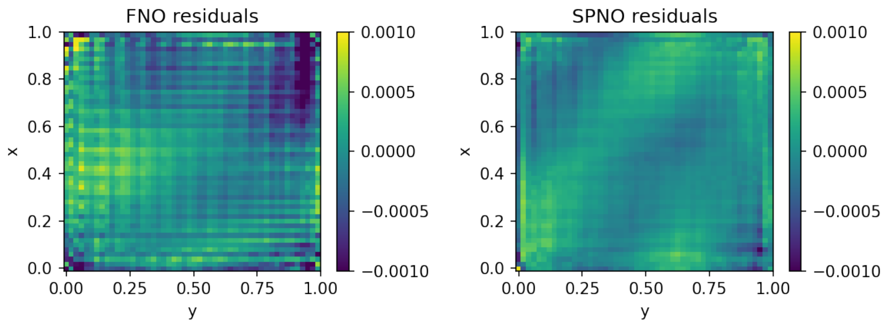
Finally we consider a elliptic differential equations with boundary problem,

$$\begin{aligned} -\varepsilon \Delta u + u_x + u_y &= f(x) \quad x \in (0,1) \,\text{and}\, y \in (0,1), \\ u(0,y) &= u(1,y) = 0, \\ u(x,0) &= u(x,1) = 0. \end{aligned} \tag{20}$$

The boundary layer of this equation exists at $x = 1$ and $y = 1$, as shown in Fig.7(a). But we also use $blocksNum = 2$. Theoretically just adding an exponential boundary layer would have a bias in the corners, but corners are not dealt with here because we think the backbone will automatically correct them. In Fig.7(b), we can clearly see that our structure acts on the whole area, not just at the boundary.



(a) predicts on both two methods



(b) residuals on both two methods

Figure 7: Performance of both FNO and SPNO on Eq.20. Take one of 1000 samples. (a) The residuals is calculated by $u_g - u_p$. u_g is the ground truth and u_p is the predicts of the model. (b) The relative loss and MSE are both presented.

4.4 Multiple distributions

The traditional FNO method cannot accommodate equations with multiple different ε as the input at the same time, because different ε will lead to different data distributions. So the correct way is to add the ε parameter as input, or put ε into the data set as a dominant feature, and the previous method is used here. Obviously our model is a multi-input form: an additional input parameter ε is required for coordinate transformation. We choose the parameter $\varepsilon \in \{2^{-2}, 2^{-3}, 2^{-4}, 2^{-5}, 2^{-6}\}$, and use it to generate 5 different solutions in one same equation and mix them together. We want to see how well the two models fit this kind of data. We can see in Fig.8 that the result of FNO does seem to be missing some data containing layer structure. In contrast, the results of SPNO are satisfactory. The residuals remain small in both stationary regions and boundary layers. It can be speculated that the layer block is not only effective at the boundary layer, but also has an interpolation effect on the prediction of the model. Of course how the interpolation is performed changes according to the mode of the layer block. Usually in these experiments, we use exponential functions as the activation function of the layer block, the purpose is to generate the corresponding decay exponential function.

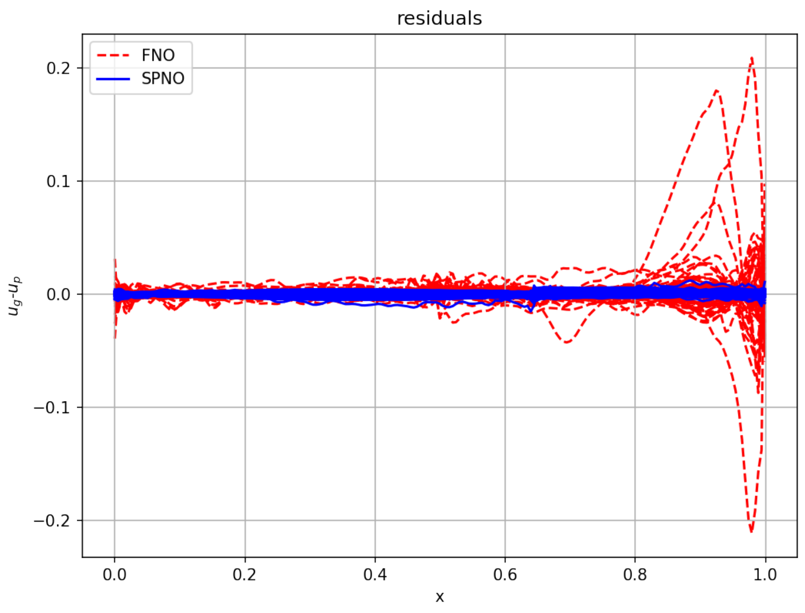


Figure 8: Residuals tested by both FNO and SPNO on Eq.17. Train and test on 5 different ε cases. FNO shows its under-fitting around the boundary layer especially the small number of samples with large variation.

4.5 Few-shot situation

In this subsection, we mainly demonstrate that our method is less sensitive at the sample size level. First of all, we have selected very few samples for training, in other words, we want to observe whether the layer blocks can provide enough physical information to make the model training successful when the amount of data decreases. We test in the simplest example Eq.17 by reducing the original 1000 samples to 100 samples.

We can see in Fig.9 there are a small number of extreme samples, which can be understood as the edge part of the data distribution. For this part of the data, usually the model cannot fit well. The reason is first, the number of extreme samples is too small. Second, in singularly perturbed differential equation, the boundary layer of the solutions of these extreme samples is very large, and the data fluctuation of the boundary layer itself is too large, which leads to the acquisition of the model at the boundary data is scarce.

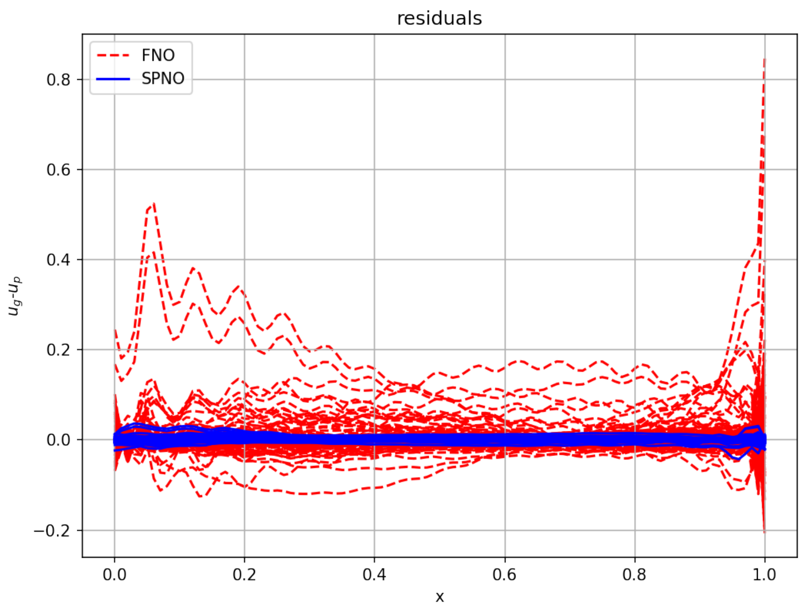


Figure 9: Residuals of FNO and SPNO on Eq.17. Trained on 100 samples and test on 100 different samples.

4.6 Metrics

The neural operator requires a large number of function samples, which is different from the point samples required by ordinary PDE solvers. For the FNO model, it can be observed that its predicted value is volatile. We argue that the presence of volatility makes the solved PDE solution misleading. Generally, when a model is perfectly trained and the error is minimized on the objective function, the actual predicted value may be unstable. For example, when the predicted value is greater than 0 in some places and less than 0 in other places, then it is not feasible to calculate the MAE because the MAE is always small at this time. To observe the volatility of the residuals, we define the mean and variance of the residuals

$$\begin{aligned} mean &= \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\hat{u}_{ij}-u_{ij}\right), \\ var &= \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\left(\hat{u}_{ij}-u_{ij}\right)-\frac{1}{M}\sum_{j=1}^{M}\left(\hat{u}_{ij}-u_{ij}\right)\right)^2. \end{aligned} \tag{21}$$

where N is the number of samples, M is the resolution of the function. We counted the mean and variance of the test set residuals for all 6 experiments, as shown in Tab.1. The closer the mean and variance are to 0, the better the model performance. It can be clearly seen that the new structure can reduce the volatility of forecast values.

Table 1 Mean and variance of residuals on test set of all experiments.

Experiment	FNO		SPNO	
	mean	var	mean	var
1D(no turning point)	1.00e-03	2.52e-05	-2.59e-05	1.96e-07
1D(turning point)	2.59e-01	1.45e+02	4.20e-03	2.04e-02
1D(initial-boundary)	-5.01e-06	2.00e-08	-8.82e-07	1.09e-09
2D	2.17e-05	1.51e-06	-2.20e-05	2.20e-07
multiple ε	6.00e-04	7.45e-05	8.30e-05	1.90e-06
few-shot	8.70e-03	9.00e-04	8.00e-04	8.70e-06

5 Conclusion

This paper mainly provides an advanced neural operator model for solving singularly perturbed differential equations. We first analyze the boundary layer function of the solution of singularly perturbed differential equations. Then we design a unique layer block structure to assist traditional neural operators or other PDE solvers to train. Experiments show that after adding layer blocks, the prediction accuracy of the model for the boundary layer can be significantly improved and the residual error can be reduced. For some special cases, such as few-shot learning, it can also outperform traditional operator networks. But it does not mean that reducing the number of samples has no effect. In fact, both models have a certain decrease in accuracy in this case. Then we test our model on multiple ε situation, found that our method is a multiple input version of FNO suitable for multiple distributions. In addition, our experiments show that our method's residuals for predicted values are more stable.

In the experiments, our method only uses the exponential layer blocks. It is our recommendation setting and do not mean that it is necessarily correct. The specific predictions are related to the actual equations. When internal layers or parabolic layers occur, only adding exponential layers may not be sufficient. Of course, adding too many layer blocks will greatly increase the parameters of the model. At this time, we suggest to appropriately reduce the parameters of the trunk network. For cases where only boundary layers are present, we still recommend $blockNum = 2$ in 1D and 2D situation.

References

[1] Roos H G, Stynes M, Tobiska L. Robust numerical methods for singularly perturbed differential equations: convection-diffusion-reaction and flow problems[M]. Springer Science & Business Media, 2008.

[2] Rackauckas C, Ma Y, Martensen J, et al. Universal differential equations for scientific machine learning[J]. arXiv preprint arXiv:2001.04385, 2020.

[3] Karniadakis G E, Kevrekidis I G, Lu L, et al. Physics-informed machine learning[J]. Nature Reviews Physics, 2021, 3(6): 422-440.

[4] Raissi M, Perdikaris P, Karniadakis G E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations[J]. Journal of Computational physics, 2019, 378: 686-707.

[5] Tawfiq L N M, Al-Abrahemee K M M. Design Neural Network to Solve Singular Pertubation Problems[J]. J. of Applied & Computational Mathematics, 2014, 3: 1-5.

[6] Liu H, Xing B, Wang Z, et al. Legendre neural network method for several classes of singularly perturbed differential equations based on mapping and piecewise optimization technology[J]. Neural Processing Letters, 2020, 51(3): 2891-2913.

[7] Yadav S, Ganesan S. SPDE-Net: Neural Network based prediction of stabilization parameter for SUPG technique[C]//Asian Conference on Machine Learning. PMLR, 2021: 268-283.

[8] Beguinet A, Ehrlacher V, Flenghi R, et al. Deep learning-based schemes for singularly perturbed convection-diffusion problems[J]. arXiv preprint arXiv:2205.04779, 2022.

[9] Li Z, Kovachki N, Azizzadenesheli K, et al. Fourier neural operator for parametric partial differential equations[J]. arXiv preprint arXiv:2010.08895, 2020.

[10] Lu L, Jin P, Pang G, et al. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators[J]. Nature Machine Intelligence, 2021, 3(3): 218-229.

[11] Goswami S, Bora A, Yu Y, et al. Physics-Informed Neural Operators[J]. arXiv preprint arXiv:2207.05748, 2022.

[12] Krishnapriyan A, Gholami A, Zhe S, et al. Characterizing possible failure modes in physics-informed neural networks[J]. Advances in Neural Information Processing Systems, 2021, 34: 26548-26560.

[13] Rosofsky S G, Huerta E A. Applications of physics informed neural operators[J]. arXiv preprint arXiv:2203.12634, 2022.

[14] Haghighat E, Bekar A C, Madenci E, et al. A nonlocal physics-informed deep learning framework using the peridynamic differential operator[J]. Computer Methods in Applied Mechanics and Engineering, 2021, 385: 114012.

[15] Lu L, Meng X, Cai S, et al. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data[J]. Computer Methods in Applied Mechanics and Engineering, 2022, 393: 114778.

[16] Wang S, Wang H, Perdikaris P. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets[J]. Science advances, 2021, 7(40): eabi8605.

[17] Kovachki N, Lanthaler S, Mishra S. On universal approximation and error bounds for Fourier Neural Operators[J]. Journal of Machine Learning Research, 2021, 22: Art. No. 290.

[18] You H, Zhang Q, Ross C J, et al. Learning Deep Implicit Fourier Neural Operators (IFNOs) with Applications to Heterogeneous Material Modeling[J]. arXiv preprint arXiv:2203.08205, 2022.

[19] Li Z, Zheng H, Kovachki N, et al. Physics-informed neural operator for learning partial differential equations[J]. arXiv preprint arXiv:2111.03794, 2021.

[20] Deng B, Shin Y, Lu L, et al. Approximation rates of DeepONets for learning operators arising from advection-diffusion equations[J]. Neural Networks, 2022.

[21] De Ryck T, Mishra S. Generic bounds on the approximation error for physics-informed (and) operator learning[J]. arXiv preprint arXiv:2205.11393, 2022.

[22] Jin P, Meng S, Lu L. MIONet: Learning multiple-input operators via tensor product[J]. arXiv preprint arXiv:2202.06137, 2022.

[23] Chen T, Chen H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems[J]. IEEE Transactions on Neural Networks, 1995, 6(4): 911-917.

[24] Lanthaler S, Mishra S, Karniadakis G E. Error estimates for DeepONets: A deep learning framework in infinite dimensions[J]. Transactions of Mathematics and Its Applications, 2022, 6(1): tnac001.

[25] Yang Y, Gao A F, Castellanos J C, et al. Seismic wave propagation and inversion with Neural Operators[J]. The Seismic Record, 2021, 1(3): 126-134.

[26] De Hoop M, Huang D Z, Qian E, et al. The Cost-Accuracy Trade-Off In Operator Learning With Neural Networks[J]. arXiv preprint arXiv:2203.13181, 2022.

[27] Simos T E, Famelis I T. A neural network training algorithm for singular perturbation boundary value problems[J]. Neural Computing and Applications, 2022, 34(1): 607-615.