DeepPropNet - A Recursive Deep Propagator Neural Network for Learning Evolution PDE Operators

Lizuo Liu, Wei Cai *

March 1, 2022

Abstract

In this paper, we propose a deep neural network approximation to the evolution operator for time dependent PDE systems over long time period by recursively using one single neural network propagator, in the form of POD-DeepONet with built-in causality feature, for a small time interval. The trained DeepPropNet of moderate size is shown to give accurate prediction of wave solutions over the whole time interval.

1 The Problem of Learning PDE Evolution Operator

Consider a second order evolution system

\[ u_{tt} = Lu + f(x,t), \]
\[ u(x,0) = u_0(x), \quad x \in \mathbb{R}^1, \]
\[ u_t(x,0) = v_0(x). \]

where \( u \) could be a scalar or vector. For a model problem, we will consider the inhomogeneous scalar wave (d’Alembert) equation

\[
\frac{\partial^2 u}{\partial t^2} - c^2 (x,t) \frac{\partial^2 u}{\partial x^2} = f(x,t), \quad x \in \mathbb{R}, \quad t \geq 0, \tag{3}
\]

with the source term \( f \) that is compactly supported on a bounded space-time domain \( Q = S \times [0,T] \) where \( S \subset \mathbb{R} \). This means that the source term \( f(x,t) \) differs from zero only on \( S \) and operates for the limited time interval \([0,T]\). When \( c(x,t) = \text{const} \), the solution to the Cauchy problem \([3]\) is given by:

\[
u(x,t) = \frac{1}{2} (u_0(x-ct) + u_0(x+ct)) + \frac{1}{2c} \int_{x-ct}^{x+ct} v_0(\xi) d\xi + \frac{1}{2c} \int_0^t \int_{x-c(t-\tau)}^{x+c(t+\tau)} f(\xi,\tau) d\xi d\tau. \tag{4}\]

The solution \( u(x,t) \) can be viewed through an evolution operator \( P(u_0, v_0, f(x,s), 0 \leq s \leq t) \), which maps the initial conditions and the source term into the solution. Learning

*Department of Mathematics, Southern Methodist University, Dallas, TX 75252. email: cai@smu.edu. Date: February 27, 2022.
such a map between functions has been actively studied recently with various types of operator learning methods, including DeepONet [4] and Fourier Neural Operator [3]. The focus of this paper is to find an efficient way to learn this evolution operator with moderate size neural network for large time $t$. For lack of a precise term, borrowing the term from quantum mechanics for the Green’s function propagator [5], we shall name the operator $P$ as the propagator for the evolution system. From [4], it is clear that if we like to train a neural network operator for large time $t$, the size of the network will grow for increasing time $t$. And, the amount of information to be input into a network will increase dramatically as $t$ grows as well. We will propose a recursive propagator formulation for the evolution operator network.

First, the solution time interval $[0, T]$ will be divided into $N$ smaller subintervals

$$t_0 = 0 < t_1 < \cdots < t_i < \cdots t_N, t_i = i\Delta t, \quad \Delta t = T/N, \quad (5)$$

and for $t_i \leq t \leq t_{i+1}$, the solution is given by the propagator with initial condition of the solution $u_i$ and its velocity $v_i = u_i$, i.e.,

$$u(x, t) = P(u_i, v_i, f(x, s), t_i \leq s \leq t), \quad (6)$$

where the initial condition $u_i$ would have been given by the propagator for the time block $t_{i-1} \leq t \leq t_i$.

The propagators in (6) will be approximated by a single neural network in the form of DeepONet structure [2] with modification for time causality, namely,

$$P(u_i, v_i, f(x, s), t_i \leq s \leq t) \sim P_{\theta}(u_i, v_i, f(x, s), t_i \leq s \leq t), 0 \leq i \leq N - 1. \quad (7)$$

Therefore, the propagator $P_\theta(u_0(x), v_0(x), f(x, t))$ will be trained to map the initial and force function data into the solution $u(x, t)$, $t_0 \leq t \leq t_1$. Moreover, this same propagator $P_\theta$ will be trained to approximate the solution for time periodic $t_1 \leq t \leq t_2$ where the initial condition at $t_1$ can be computed with the propagator for the previous time interval $[t_0, t_1]$. This procedure will be used recursively until we have trained the same propagator for the last time interval $[t_{N-1}, t_N]$. Taken all together, we arrive at a propagator neural network for the whole time interval $[0, T]$ where the building block is the single propagator of moderate size $P_\theta(u_0(x), v_0(x), f(x, t), t_0 \leq t \leq t_1)$. By controlling the size of the $\Delta t$, the size of this propagator can be easily controlled for efficiency as well as accuracy. As the evolution PDE system has to observe the causality of the physical system, the DeepONet framework will be modified to include the causality, a previously proposed causality DeepONet in the study of dynamics system for modeling building response to seismic waves [4] will be used for this purpose.

The rest of the paper is organized as follows. In section 2 we will review the DeepONet [4] with time causality and extension with proper orthogonal decomposition (POD) approach for efficient treatment of spatial dependence of the solution. Section 3 will give the algorithmic outline of the DeepPropNet and numerical results of the DeepPropNet will be presented in Section 4. Finally, Section 5 gives a conclusion and some future work.

2 DeepONet with time causality and spatial POD

A causality DeepONet was proposed in [4] to handle the time causality in dynamics system and was shown to be very effective to predict the seismic response of building. Here, we will just present the final form and for details please refer to [4].
**Causality-DeepONet:** A DNN representation of an operator $G(f(t))$ for any continuous function $f(t)$ with retarded response for $t \in K_2 = [0, T] \subset \mathbb{R}$ is given as

$$G(f)(t) \sim \sum_{k=1}^{N} \sum_{i=1}^{M} c^k_i \sigma_b \left( \sum_{j=1}^{\left\lceil \frac{t}{h} \right\rceil} \xi^k_{i,m} - \left\lceil \frac{t}{h} \right\rceil + j \right) f (s_j) \cdot \sigma_trk (\omega_k \cdot t + \zeta_k),$$  \hspace{1cm} (8)

where $\{s_j\}_{j=1}^{m} \subset K_1 = [0, T] \subset X$, coefficients $c^k_i, \xi^k_{i,j}, \omega_k, \zeta_k$ - all independent of continuous functions $u \in V \subset C(K_1)$ and $t$.

To handle the spatial dependence of solution $u(x, t)$ for the evolution system, we will adopt the idea of the POD-DeepONet\cite{6, 7} which assumes there is a set of global basis for the targeted output which could be found by SVD, and the trunk net will be replaced by these basis, then the neural network is learning the mapping between values at sensors to the singular values if we consider using SVD as an example. Thus it may lose the ability to approximate arbitrary values of the operator in the computational domain given the values at sensors. The philosophy of POD-DeepONet is to find a set of basis globally, thus compress the memory and computation needed, which is the crux for learning the operator of the high dimensional problem($d \gg O(1)$).

Lu et al.\cite{7} discussed the POD-DeepONets and proposed a time marching scheme for Fourier Neural Operator specifically to reduce the dimension of output, and they also proposed the modified DeepONet with feature expansion by feeding historical states of signal to trunk net as features. Bhattacharya et al.\cite{6} proposed that they reduce the dimension of input and output by PCA and then learning the mapping between the reduced space by a neural network, and they also discussed the approximation quality needed to get good mappings. Meuris et al.\cite{1} presented a procedure that learning the data-driven basis functions harnessing the DeepONet machinery and the learned basis function will be used by classical methods as custom basis to achieving high accuracy for computation of arbitrary complex domain.

As for a problem with causality, The POD-DeepONet’s philosophy guides us either do the SVD for the whole outputs regardless the difference of temporal variables and spatial variables, or do specific SVD timestep by timestep to keep the causality for which we need to keep the basis for all time steps, or consider solution at each time step for each case as an independent target and find a common basis w.r.t spatial dimensions for all time steps & all cases. These ideas either burns the high memory cost, or destroy the causality in a brutal-force way. Therefore, a modification of POD-DeepONet for problem with causality is crucial. We follow the idea of POD-DeepONet but only construct the basis of spatial domain explicitly and utilize the causality DeepOnet to handle the temporal-dependent coefficients of each spatial basis.

In the following, we consider the 1d wave equation (3) and assume the right hand side has form

$$f(x, t) = a_0(t) + \sum_{n=1}^{N} a_n(t) \cos(2n\pi x) + b_n(t) \sin(2n\pi x), \hspace{1cm} (9)$$

and the solution has a similar form

$$u(x, t) = \psi_0(t) + \sum_{m=1}^{M} \psi_m(t) \cos(2m\pi x) + \phi_m(t) \sin(2m\pi x), \hspace{1cm} (10)$$

Next, we modify the Causality-DeepONet (8) by

$$\mathcal{G}(\vec{f})(x, t) \sim \sigma_{br} (\vec{f}) \odot \sigma_{trk}(t) \cdot \sigma_{basis}(x), \hspace{1cm} (11)$$
where $\odot$ is the elementwise multiplication, $\cdot$ is the inner product,

$$
\overrightarrow{f}(s) = [a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)],
$$

$$
\overrightarrow{\sigma}_{br,i}(\overrightarrow{f}) = \sigma_b \left( \sum_{j=1}^{[\frac{\pi}{\xi}]} \xi_{i,m}^0 + j a_0(s_j) \right)
+ \sum_{n=1}^{N} \sum_{j=1}^{[\frac{\pi}{\xi}]} \left( \xi_{n,a}^n + j a_n(s_j) + \xi_{n,b}^n + j b_n(s_j) \right),
$$

$$
\overrightarrow{\sigma}_{trk,i}(t) = \sigma_{trk}(\omega_i \cdot t + \zeta_i),
$$

$$
\overrightarrow{\sigma}_{basis}(x) = \{1, \cos(2\pi x), \sin(2\pi x), \ldots, \cos(2M\pi x), \sin(2M\pi x)\}.
$$

The activation functions of $\sigma_b$ and $\sigma_{trk}$ are ReLU. Note the modes number $M$ in the CPOD-DeepONet (Causality POD-DeepONet) need to be greater than or equal to the number of given modes $N$ of right hand side in equation (9). The algorithm of the CPOD-DeepONet is shown in algorithm 1. To contain the memory efficiency and following the fact that equation (10) has separation of variables, we will do outer product rather than computing point by points as shown in the last line of algorithm. This is one of the crux to handle higher dimensional problem.

**Algorithm 1:** The algorithm of Causality POD-DeepONet

| Input | $t$, $x$, $[a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)]$ |
|-------|-----------------------------------------------------|
| Output | $u(x,t)$ |

$y_t \leftarrow CDeepONet(t, [a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)])$; /* $CDeepONet$ is the causality DeepONet. The output $y_t$ has size of $[\text{batchsize} \times N_t \times \text{NBx}]$ */

$y_x \leftarrow [1, \cos(2\pi x), \sin(2\pi x), \ldots, \cos(2M\pi x), \sin(2M\pi x)]$; /* The output $y_x$ has size of $[\text{batchsize} \times N_x \times \text{NBx}]$ */

$u(x,t) \leftarrow \text{Einsum('bxn,btn \rightarrow bxt', } y_x, y_t)$; /* The Einsum is the Einstein summation convention, ('bxn,btn \rightarrow bxt') means the index change in the Einstein summation convention. */

**Loss function** Given batch size $N$ for training process and the total number of the test records $N$, the loss function is defined as

$$
\text{Loss}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N_t} \sum_{k=1}^{N_x} \left( G \left( \overrightarrow{f} \right)(x_k, t_j) - y_{ijk} \right)^2,
$$

where $N_t$ is the number of time step and $N_x$ is the number of points on $x$ direction.

To evaluate the training process, the mean of the relative L2 error is considered

$$
\text{Loss}_2(\theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{\left\| G \left( \overrightarrow{f} \right)(x, t) - u_i(x, t) \right\|_2}{\left\| u_i(x, t) \right\|_2}.
$$
The relative L2 error in a complete epoch is defined as

\[
\mathcal{L}_{\text{train}}^R = \frac{1}{B} \sum_{k=1}^{B} \frac{1}{N} \sum_{i=1}^{N} \frac{\| \mathcal{G} \left( \vec{f}_i \right) (x, t, \theta^{(k)}) - u_{ik} (x, t) \|_2}{\| u_i (x, t) \|_2},
\]

(17)

where \( B \) is the number of batches, \( \theta^{(k)} \) means the parameters of neural network at \( k \)-th batch.

Similarly, we define the relative L2 error for the testing dataset

\[
\mathcal{L}_{\text{test}}^R = \frac{1}{N} \sum_{i=1}^{N} \frac{\| \mathcal{G} \left( \vec{f}_i \right) (x, t) - u_i (x, t) \|_2}{\| u_i (x, t) \|_2}.
\]

(18)

Note the \( N \) is the total number of test cases.

- **Numerical Performance of Causality DeepONet with POD**

  **Case 1: Constant wave-speed case**

  In this test, we assume the wave speed is 2, and the right hand side \( f (x) \) satisfies

  \[
  f (x, t) = \sum_{i=0}^{2} c_i t + \sum_{i=1}^{10} 2(c_i^2 - 1) \pi^2 \left( a_i \cos(2i\pi t) \cos(2i\pi x) + b_i \sin(2i\pi t) \sin(2i\pi x) \right),
  \]

  (19)

  where \( a_i, b_i, c_i \) are randomly sampled from \([0, 1] \). The exact solutions are from integrating the formula based on Duhamel’s method. There are 21 basis based on [19] thus, the input size of the causality DeepONet should be 21 \( \times \) \( N_t \), where \( N_t \) is the number of time steps. We set \( N_t = 400 \) for all other cases in this paper. We build the branch part and trunk part of the causality DeepONet as 4-layer neural network with 128 hidden neurons each layer and 4-layer neural network with 100 hidden neurons each layer respectively, both of whose outputs dimension is 500 and shrinking to dimension \( N_x \times N_t \) by inner product with the \( x \)-basis. The learning rates during training including the following cases are \( 10^{-4} \). The batch size we choose for all the testing case is 20. The results trained 500 epochs with 1000 cases are shown in Figure 1.

![Figure 1: Random testing case after 500 epochs training for the simple case. The maximum relative error is 5%.

- **Case 2: Variable wave-speed case**

  In this case we consider the case the wave speed is a function of \( x, t \)

  \[
  c (x, t) = \sqrt{\cos \left( 2n\pi \left( t + x \right) \right) + 1},
  \]

  (20)
and the exact solution is given by

\[ u(x, t) = \sum_{m=1}^{M} c_m \cos(2m\pi (t + x)), \] (21)

thus, the corresponding right hand side for (3) will be

\[ f(x, t) = \sum_{m=1}^{M} c_m (2m\pi)^2 [\cos(2\pi (m + n) t) \cos(2\pi (m + n) x) + 
\quad \sin(2\pi (m + n) t) \sin(2\pi (m + n) x) 
\quad + \cos(2\pi (n - m) t) \cos(2\pi (n - m) x) 
\quad - \sin(2\pi (n - m) t) \sin(2\pi (n - m) x)] . \] (22)

The inputs for the CPOD-DeepONet are the coefficients of basis function of \( x \) in (22), since there are \( 4M \) basis, the input size of the causality DeepONet should be \( 4M \times N_t \). The causality DeepONet is as the same shape as in Case 1 but with different input dimensions. In this case we assume \( M = 10 \) and \( n = 10 \) with 400 equal spaced time steps, and there are 400 random sampled points on \( x \)-direction for each time step. We have similar results as shown in Figure 2. It should be mentioned that the trained CPOD-DeepONet can in fact be used to predict solution outside the trained spatial domains as shown in Fig. 3.

![Figure 2: Random testing case after 500 epochs training for the variable wave speed case. The maximum relative error is 4.2 ± 2.3%.](image)

![Figure 3: Random testing case after 500 epochs training for the variable wave speed case with random large domain on \( x \in [-10, 10] \). The maximum relative error is 4.0 ± 1.4%.](image)
3 A Recursive DeepPropNet for Learning Evolution PDE Operators

Following the semi-group formulation of evolution PDEs, the DeepPropNet computes block by block along the time direction recursively. The initial conditions for the PDEs will be included together with the force functions for the time block as an input for DeepPropNet to assure the well-posedness of the problem. And, the DeepPropNet is then constructed recursively with a Causality DeepONet with POD to learn the PDE evolution operator over long time interval.

To simplify notation, we denote the initial conditions on the solution $u$ and its velocity $v$ as

$$u_d(x) = [u_0(x), v_0(x), u_1(x), v_1(x), \ldots, u_n(x), v_n(x)], \quad (23)$$

and the DeepPropNet $P_\theta$. Following the notations of Causality DeepONet with POD in equation (11), we define the DeepPropNet as

$$P_\theta(\vec{f})(x,t) \sim \left\{ \vec{\sigma}_{br,v}(\vec{u}_d) \odot \vec{\sigma}_{trk,v}(t) + \vec{\sigma}_{br,c}(\vec{f}) \odot \vec{\sigma}_{trk,c}(t) \right\} \cdot \vec{\sigma}_{basis}(x), \quad (24)$$

where $\vec{\sigma}_{br,c}(\vec{f})$ is the branch net (12), $\vec{\sigma}_{trk,c}(t)$ is the corresponding trunk net as shown in equation (13). Likewise, $\vec{\sigma}_{br,v}(\vec{u}_d)$ and $\vec{\sigma}_{trk,v}(t)$ are the trunk net and branch net of the vanilla DeepONet,

$$\vec{\sigma}_{br,v,i}(\vec{u}_d) = \sigma_b \left( \sum_{n=0}^{N} \sum_{j=1}^{N} \xi_{i,j} u_n(s_j) + b_i \right), \quad (25)$$

the corresponding $i$-th output of trunk net is

$$\vec{\sigma}_{trk,v,i}(t) = \sigma_{trk}(\omega_i \cdot t + \zeta_i). \quad (26)$$

The notation is abused to some extent in above equations, but the $\sigma_{trk}$’s and $\sigma_b$’s on the right hand side means different neural networks. The definition of $\vec{\sigma}_{basis}$ is as the same as (14). Note the summation of $\vec{\sigma}_{br,v}(\vec{u}_d) \odot \vec{\sigma}_{trk,v}(t)$ and $\vec{\sigma}_{br,c}(\vec{f}) \odot \vec{\sigma}_{trk,c}(t)$ in the equation (24) is inspired from the superposition of linear waves from different sources and other format could be explored for nonlinear problems.

- **Recursive Formulation of DeepPropNet:** To predict the wave field in the time block $[t_0, t_1]$ at beginning, the Deep Propagator DeepPropNet is to learn the mapping

$$P_\theta : [u_d(x_j), f(x_\xi, s), t_0 \leq s \leq t_1] \mapsto u(x,t) \quad t_0 \leq t \leq t_1, x \in \mathbb{R}^n. \quad (27)$$

Once the Deep Propagator is learned, the wave field in the next time block $[t_1, t_2]$ could be predicted by the Deep Propagator by using the initial propagator as follows

$$u(x,t) = P_\theta \left[ P_\theta(x_j, t_1), P_\theta(x_j, t_1), f(x_\xi, s), t_1 \leq s \leq t_2 \right] \mapsto u(x,t) \quad t_1 \leq t \leq t_2, x \in \mathbb{R}^n, \quad (28)$$

where the initial condition is replaced by the prediction of DeepPropNet at time $t_1$.

The resulting propagator for the time block $[t_1, t_2]$ then will be again to be used to provide the initial condition for $t = t_2$ and the same initial propagator network for the time period $[t_0, t_1]$. This procedure can be carried on recursively until the whole time
period \([0,T]\) is covered, a global propagator network is thus obtained. Since it is like the initial propagator solver tracks the waves and propagates with the solutions along time direction and the propagator itself is a deep neural network, we call it Deep Propagator.

The schematics of the Deep Propagator DeepPropNet are shown in Algorithm 2. This is another crux to solve high dimensional problem. As illustrated in section 1, the input size could explode since the global dependence of source term, by solving it block by block recursively combining with the memory-efficient trick from separation of variables, learning high dimensional evolution operators is manageable now.

**Loss function** Similar to the loss function in section 2, we will define the loss function of DeepPropNet block by block in time. Given a batch size \(N\) for training process, number of blocks \(N_b\) and the total number of the test records \(N_t\), the loss function is defined as

\[
\text{Loss} (\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{n=1}^{N_t} \sum_{j=1}^{N_t,n+1} \sum_{k=1}^{N_x} (G (\bar{f} (x_k, t_j)) - y_{ijk})^2 ,
\]

where the \(n\)-th block starts from \(t_{N_t,n}\) but ends in \(t_{N_t,n+1}\) and \(N_x\) is the number of points on \(x\) direction.

Likewise, the mean of the relative L2 error is considered as the relative L2 error in the total time-spatial domain

\[
\text{Loss}_2 (\theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{\| \sum_{i=1}^{N_b} a_i z^i \| u_i (x, t) \|}{\Omega_i},
\]

where \(\Omega_i\) is the \(i\)-th block of time and spatial domain.

**Algorithm 2:** The schematics of Deep Propagator

\[
\begin{align*}
\text{Input} : & \quad t: \text{Time step} \\
& \quad x: \text{Spatial Coordinates} \\
& \quad u_d (x): \text{Initial conditions} \\
& \quad [a_0 (s), a_1 (s), b_1 (s), \ldots, a_N (s), b_N (s)]: \text{Coefficients of right hand side} \\
& \quad N: \text{Number of blocks at time domain.} \\
\text{Output:} & \quad u (x, t) \\
& \quad u_0 (x) \leftarrow u_d (x) \text{ for } (i = 0; i < N; i++) \\
& \quad \quad \quad \quad \quad \text{u} (x, t) \leftarrow \text{DeepPropaNet} (t, u_0 (x_j); [a_0 (s), a_1 (s), b_1 (s), \ldots, a_N (s), b_N (s)]); \\
& \quad \quad \quad \quad \quad u_0 (x) \leftarrow u (x, t_{i+1}); \\
\end{align*}
\]

4  **Numerical Results of DeepPropNet**

The equation and right hand sides are as the same as the non-constant wave speed case \([20] \sim [22]\), but we split 5 equal sized blocks in time domain and for each block, there are 80 sampled time locations for training. The causality DeepONet is as the same shape as in Case 2 but with different input dimensions since the dependence of time steps shrinks. The vanilla DeepONet is a composition of a 4-layer fully connected neural network with 128 hidden neurons each layer as branch net and a 4-layer fully connected neural network
**Algorithm 3:** The algorithm of DeepPropaNet

\[ t: \text{Time step} \]
\[ x: \text{Spatial Coordinates} \]
\[ u_d(x): \text{Initial conditions} \]
\[ [a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)]: \text{Coefficients of right hand side} \]

**Input:**

\[ y_t \leftarrow \text{CDeepONet} \left( t, [a_0(s), a_1(s), b_1(s), \ldots, a_N(s), b_N(s)] \right); /* The output \( y_t \)
\[ y_i \leftarrow \text{DeepONet} \left( t, u_d(x_j) \right); /* \text{DeepONet is the vanilla DeepONet. The output} \( y_i \)
\[ y_x \leftarrow [1, \cos(2\pi x), \sin(2\pi x), \ldots, \cos(2M\pi x), \sin(2Mx)]; /* \text{The output} \( y_x \)
\[ u(x, t) \leftarrow \text{Einsum}(\text{bxn,btn} \rightarrow \text{bxt}, y_x, y_t + y_i) \]

with 100 hidden neurons each layer as trunk net, both of whose output dimensions are also 500. The initial data for each block will be given by true data. Figure 5 shows the prediction of the exact solution up to time \( t = 1 \) with a maximum relative error of 3.4 \( \pm \) 2.0%.

**Training with DeepPropNet Prediction as Initial Condition at \( t_i \).** There are two strategy to provide initial conditions for each block, either provide the exact solution \( u(x, t_i) \) or the prediction \( P_\theta [u_0(x_j), v_0(x_j), f] (x, t_1) \). We compare the relative L2 error evolution of these 2 different process and as shown in Figure 6, they have similar loss convergence. The results are shown in Figure 4 and Figure 5 respectively. This is a natural test about the capability of the interpolation in training domain of the neural network, but it offers a convenience that there is no further need to prepare exact data on grids for initial conditions of each time blocks.

Figure 4: One of the random testing cases after 500 epochs training for the variable wave speed case with DeepPropNet whose initial condition is given by the prediction in \( x \in [-0.5, 1] \). The maximum relative error in the whole domain is 3.2 \( \pm \) 1.9%. (Left) Exact, (Middle) Prediction, (Right) Relative Error.

**5 Conclusion and Future works**

In this paper, we proposed the recursive way to construct DeepPropNet - a DNN propagator for evolution system over large time period by using a single building block propagator over a small time period, thus reducing the overall complexity and size of the neural
Figure 5: One of the random testing cases after 500 epochs training for the variable wave speed case with DeepPropNet whose initial condition is given by true data in $x \in [-0.5, 1]$. The maximum relative error in the whole domain is $3.4 \pm 2.0\%$. (Left) Exact, (Middle) Prediction, (Right) Relative Error.

(a) The case using prediction $P_{\theta} [u_0 (x_j), v_0 (x_j), f] (x, t_1)$ as initial condition for each time block.

(b) The case using exact solution $u (x, t_i)$ for initial condition for each time block.

Figure 6: Relative L2 error evolution during training
network required. For the design of the DeepPropNet, we also extended the Causality DeepONet with POD with specific basis to alleviate the memory burden for large spatial variables. By separately handling the spatial and temporal domain, we gain not only the memory efficiency but also the training boost. The preliminary numerical results have shown the feasibility of this recursive DeepPropNet in predicting the time evolution of wave propagations.

The proposed DeepPropNet here is based on a supervised learning approach where the data can be generated by a separate numerical methods or observation data or analytical solution when available. In theory, we could also use an unsupervised learning procedure to train the DeepPropNet by using the following residue of the PDEs as the loss function.

\[
\mathcal{L}_{\text{loss}} = \int_{\Omega} \left| \mathcal{L}_t P_\theta [u_0, v_0, f] - \mathcal{L}_x P_\theta [u_0, v_0, f] - f \right|^2 \, dx \, dt.
\]  

(31)

It is also natural to extend this framework to learn high frequency problem, since the spatial oscillating terms will be handled explicitly by the basis, but the temporal oscillating terms could be handled by the Causality DeepONet, based on our experience in the work [4].

Future work will be conducted on more complex evolution systems, the unsupervised training, training over partial time domain, highly oscillating problems and higher dimensional problems as well as initial boundary value problems.

References

[1] Brek Meuris, Saad Qadeer, and Panos Stinis. Machine-learning custom-made basis functions for partial differential equations. [arXiv:2111.05307 [cs, math]], November 2021.

[2] Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence. 2021 Mar;3(3):218-29.

[3] Li Z, Kovachki NB, Azizzadenesheli K, Bhattacharya K, Stuart A, Anandkumar A. Fourier Neural Operator for Parametric Partial Differential Equations. International Conference on Learning Representations 2020 Sep 28.

[4] L.Z. Liu, K. Nath, W, Cai, Causality DeepONet for Nonlinear Operators for Building Seismic Wave Responses, manuscript, December, 2021.

[5] R. Shankar, Principles of Quantum Mechanics, Kluwer Academic/Plenum Publisher, 1994.

[6] Koushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model Reduction And Neural Networks For Parametric PDEs. The SMAI journal of computational mathematics, 7:121–157, July 2021.

[7] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. [arXiv:2111.05512 [physics]], November 2021.