Contents lists available at ScienceDirect



Journal of Computational Physics



journal homepage: www.elsevier.com/locate/jcp

Energy-Dissipative Evolutionary Deep Operator Neural Networks

Jiahao Zhang^{a,1}, Shiheng Zhang^{a,1}, Jie Shen^{a,*}, Guang Lin^{a,b,*}

^aDepartment of Mathematics, Purdue University, 150 N. University Street, West Lafavette, IN 47907-2067, USA ^bSchool of Mechanical Engineering, Purdue University, 585 Purdue Mall, West Lafayette, IN 47907-2067, USA

ARTICLE INFO

Evolutionary Neural Networks

ABSTRACT

Energy-Dissipative Evolutionary Deep Operator Neural Network is an operator learning neural network. It is designed to seek numerical solutions for a class of partial differential equations instead of a single partial differential equation, such as partial differential equations with different parameters or different initial conditions. The network consists of two sub-networks, the Branch net, and the Trunk net. For an objective operator G, the Branch net encodes different input functions u at the same number of sensors y_i , i = $1, 2, \dots, m$, and the Trunk net evaluates the output function at any location. By minimizing the error between the evaluated output q and the expected output $\mathcal{G}(u)(y)$, DeepONet generates a good approximation of the operator \mathcal{G} . In order to preserve essential physical properties of PDEs, such as the Energy Dissipation Law, we adopt a scalar auxiliary variable approach to generate the minimization problem. It introduces a modified energy and enables unconditional energy dissipation law in the discrete level. By taking the parameter as a function of the time t variable, this network can predict the accurate solution at any further time with feeding data only at the initial state. The data needed can be generated by the initial conditions, which are readily available. In order to validate the accuracy and efficiency of our neural networks, we provide numerical simulations of several partial differential equations, including heat equations, parametric heat equations, and Allen-Cahn equations.

© 2023 Elsevier Inc. All rights reserved.

1. Introduction

Operator learning is a popular and challenging problem with potential applications across various disciplines. The opportunity to learn an operator over a domain in Euclidean spaces[1] and Banach spaces[2] opens a new class of problems in neural network design with generalized applicability. In application to solve partial differential equations(PDEs), operator learning has the potential to predict accurate solutions for the PDE by acquiring extensive prior

^{*}Corresponding authors.

e-mail: shen7@purdue.edu (Jie Shen), guanglin@purdue.edu (Guang Lin)

¹These two authors contributed equally to this work.

knowledge [3, 4, 5, 6, 7, 8, 9, 10, 11]. In a recent paper [12], Lu, Jin, and Karniadakis proposed an operator learning method with some deep operator networks, named as DeepONets. It is based on the universal approximation theorem [13, 14, 15]. The goal of this neural network is to learn an operator instead of a single function, which is usually the solution of a PDE. For any operator \mathcal{G} on a domain Ω , we can define \mathcal{G} as a mapping from $\Omega^* \to \Omega^*$ with $\mathcal{G}(u)(y) \in \mathbb{R}$ for any $y \in \Omega$. $\mathcal{G}(u)(y)$ is the expected output of the neural network, which is usually a real number. The objective of the training is to obtain an approximation of \mathcal{G} , where we need to represent operators and functions in a discrete form. In practice, it is very common to represent a continuous function or operator by the values evaluated at finite and enough locations $\{x_1, x_2, \dots, x_m\}$, which is called "sensors" in DeepONet. The network takes $[u(x_1), u(x_2), \dots, u(x_m)]$ and y as the input. The loss function is the difference between the output q and the expected output $\mathcal{G}(u)(y)$. Generally, there are two kinds of DeepONet, Stacked DeepONet, and Unstacked DeepONet. The Stacked DeepONet consists of p branch networks and one trunk network. The number of the Trunk networks of the Unstacked DeepONet is the same as the DeepONet, but the Unstacked DeepONet merges all the p branch networks into a single one. An Unstacked DeepONet combines two sub-networks, Branch net, and Trunk net. The Branch net encodes the input function u at some sensors, $\{x_i \in \Omega \mid i = 1, \dots, m\}$. The output of the Branch net consists of p neurons, where each neuron can be seen as a scalar, $b_i = b_i(u(x_1), u(x_2), \cdots, u(x_m)), j = 1, 2, \cdots, p$. The Trunk net encodes some evaluation points $\{y_k \in \Omega | k = 1, \dots, n\}$, while the output also consists of p neurons and each neuron is a scalar $g_i = g_i(y_1, y_2, \dots, y_n)$, $j = 1, 2, \dots, p$. The evaluation point y_i can be arbitrary in order to obtain the loss function. The number of neurons at the last layer of the Trunk net and the Branch net is the same. Hence, the output of the DeepONet can be written as an inner product of (b_1, b_2, \dots, b_p) and (g_1, g_2, \dots, g_p) . In other words, the relationship between the expected output and the evaluated output is $\mathcal{G}(u)(y) \approx \sum_{j=1}^{p} b_j g_j$. The DeepONet is an application of the Universal Approximation Theorem for Operator, which is proposed by Chen & Chen [16]:

Theorem 1.1 (Universal Approximation Theorem for Operator). Suppose that Ω_1 is a compact set in X, X is a Banach Space, V is a compact set in $C(\Omega_1)$, Ω_2 is a compact set in \mathbb{R}^d , σ is a continuous non-polynomial function, \mathcal{G} is a nonlinear continuous operator, which maps v into $C(\Omega_2)$, then for any $\epsilon > 0$, there are positive integers M, N, m, constants $c_i^k, \zeta_k, \xi_{ij}^k \in \mathbb{R}$, points $\omega_k \in \mathbb{R}^n, x_j \in K_1, i = 1, \dots, M, k = 1, \dots, N, j = 1, \dots, m$, such that

$$|\mathcal{G}(u)(y) - \sum_{k=1}^{N} \sum_{i=1}^{M} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u\left(x_j\right) + \theta_i^k \right) \cdot \sigma \left(\omega_k \cdot y + \zeta_k\right) | < \epsilon$$

holds for all $u \in V$ and $y \in \Omega_2$.

For any time-dependent PDE, the training data is the form of $(u, y, \mathcal{G}(u)(y))$, where u in the discrete form can be represented as $[u(x_1), u(x_2), \dots, u(x_m)]$ in the neural network. In the original paper, they used the classic FNN[17] as the baseline model. For dynamic systems, various network architectures are used, including residual networks[18], convolutional NNs(CNNs)[19, 20], recurrent NNs(RNNs)[21], neural jump stochastic differential equations[22] and neural ordinary differential equations[23]. The training performance is very promising. It predicts accurate solutions of many nonlinear ODEs and PDEs, including the simple dynamic system, gravity pendulum system, and diffusion-reaction system. However, the training data need to be generated at each time step, so it is very expensive to train the network. For a lot of initial value problems, there is no any information of u(x, t) except t = 0. It is very natural to raise a question: *Can we learn an operator of a kind of time-dependent PDEs with only initial conditions?*

Inspired by the Evolutionary Deep Neural Network(EDNN)[24], it is more convenient to learn an operator at a fixed time instead of an operator with not only spatial variables but also a time variable. With the loss of generality, we can take the time variable *t* to be 0 in initial value problems. Once obtained the operator at the initial time, many traditional numerical methods can be used to update the solution. More specifically, assuming that the initial condition operator has been trained well, we can consider the parameters of the Branch net and the Trunk net as a function with respect to the time variable as shown in Figure 1. More specifically, for a given initial value problem,

$$\begin{cases} \frac{\partial u}{\partial t} = s(u) \\ u(x,0) = f(x), \quad x \in \omega \end{cases}$$
(1)

the objective is to approximate the operator $\mathcal{G} : u \mapsto \mathcal{G}(u)$. The input is $([u(x_1), u(x_2), \dots, u(x_m)], y, \mathcal{G}(u)(y))$, where $\{x_1, x_2, \dots, x_m\}$ are the sensors and $\mathcal{G}(u)(y) = f(y)$. The training process at the initial step is the same as the DeepONet,



(b) Unstacked EDE-DeepONet

Fig. 1: Energy-Dissipative Evolutionary Deep Operator Neural Network. The yellow block represents input at sensors and the blue block represents subnetworks. The green blocks represent the output of the subnetworks and also the last layer of the EDE-DeepONet. The difference between the stacked and unstacked EDE-DeepONet is the number of Branch nets. In the right minimization problem, the energy term r^2 can be shown to be dissipative, i.e. $(r^{n+1})^2 \le (r^n)^2$, where $\mathcal{J}(\gamma_1, \gamma_2) = \frac{1}{2} \left\| \sum_{k=1}^p \frac{\partial g_k(W_1^n)}{\partial W_1^n} \gamma_1 \boldsymbol{b}_k(W_2^n) + \sum_{k=1}^p g_k(W_1^n) \frac{\partial \boldsymbol{b}_k(W_2^n)}{\partial W_2^n} \gamma_2 - \frac{r^{n+1}}{\sqrt{E(\boldsymbol{u}^n)}} \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}^n) \right\|_2^2$.

so we can use the same architecture to train the initial condition operator. The output of the Branch net can be written as $\boldsymbol{b} = \boldsymbol{b}(u(x_1, 0), u(x_2, 0), \dots, u(x_{m_1}, 0)) = \boldsymbol{b}'(x_1, x_2, \dots, x_{m_1}; W_1)$, where W_1 are the parameters in the Branch net. The output of the Trunk net can be written as $\boldsymbol{g} = \boldsymbol{g}(y; W_2)$, where W_2 are the parameters in the Trunk net. Once trained well, we will regard the parameters as a function of t and W_1 , W_2 as the initial conditions of $W_1(t)$ and $W_2(t)$. By the architecture of the Unstacked DeepONet, we can write the solution at initial time $t_0 = 0$ as

$$u(x, t_0) \approx \sum_{j=1}^{p} b_j g_j = \boldsymbol{b}^T \boldsymbol{g} \text{ for any given initial condition } f(x)$$
(2)

We do not need any more data to obtain the approximation of $u(x, t_1)$. $u(x, t_1)$ should be consistent with $W_1(t_1)$ and $W_2(t_1)$. With the idea of the numerical solver for PDEs, it is easy to obtain $W_1(t_1)$ and $W_2(t_1)$ if $\frac{\partial W_1}{\partial t}$ and $\frac{\partial W_2}{\partial t}$ are known. The time derivative of the solution u can be written by a chain rule:

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial W} \frac{\partial W}{\partial t}$$
(3)

where W consists of W_1 and W_2 . $\frac{\partial W}{\partial t}$ can be solved by a least square problem. Once we get $\frac{\partial W}{\partial t}$, we can use any traditional time discretization schemes to get W^{n+1} with W^n .

The choice of the traditional time discretization scheme is dependent on the specific problem. The Euler or Runge–Kutta methods are commonly used in the evolutionary network. We are going to introduce a method with unconditional energy dissipation, which is the Energy-Dissipative Evolutionary Deep Operator Neural Network(EDE-DeepONet). Many kinds of PDEs are derived from basic physical laws, such as Netwon's Law, Conservation Law and Energy Dissipation Law. In many areas of science and engineering, particularly in the field of materials science, gradient flows are commonly employed in mathematical models[25, 26, 27, 28, 29, 30, 31, 32]. When approximating the solution of a certain PDE, it is desirable to satisfy these laws. We consider a gradient flow problem,

$$\frac{\partial u}{\partial t} = -\frac{\delta E}{\delta u},\tag{4}$$

where E is a certain free energy functional. Since the general explicit Euler method does not possess the unconditionally dissipative energy dissipation law, we applied a scalar auxiliary variable(SAV) method[33] to generate the required least square problem. It introduces a new modified energy and the unconditionally dissipative modified energy dissipation law is satisfied for each iterative step. SAV method has been applied to solve plenty of PDEs with thermodynamically consistent property. It is robust, easy to implement and accurate to predict the solution. Introducing this method to neural network helps us explore how to combine neural network models and physical laws. The objectives of this article is:

- Designing an operator learning neural network without data except the given information.
- Predicting solutions of parametric PDEs after a long time period.
- Keeping energy dissipative property of a dynamic system.

Our main contributions are:

- Constructing an evolutionary operator learning neural network to solve PDEs.
- Solving a kind of PDEs with different parameters in a single neural network.
- Introducing the modified energy in the neural network and applying SAV algorithm to keep the unconditionally modified energy dissipation law.
- Introducing an adaptive time stepping strategy and restart strategy in order to speed the training process.

The organization of this paper is as follows: In Section 2, we introduce the Evolutionary Deep Operator Neural Network for a given PDE problem. In Section 3, we consider the physics law behind the gradient flow problem and apply the SAV method to obtain the energy dissipation law. We proposed a new architecture for neural network, EDE-DeepONet. In Section 4, we presented two adaptive time stepping strategies, where the second one is called restart in some cases. In Section 5, we generally introduced the architecture of the EDE-DeepONet. In Section 6, we implement our neural network to predict solutions of heat equations, parametric heat equations, and Allen-Cahn equations to show the numerical results.

2. Evolutionary Deep Operator Neural Network

Consider a general gradient flow problem,

$$\frac{\partial u}{\partial t} + \mathcal{N}_{\mathbf{x}}(u) = 0 \tag{5}$$
$$u(\mathbf{x}, 0) = f(\mathbf{x})$$

where $u \in \mathbb{R}^l$, $\mathcal{N}_x(u)$ can be written as a variational derivative of a free energy functional E[u(x)] bounded from below, $\mathcal{N}_x(u) = \frac{\delta E}{\delta u}$. The first step is to approximate the initial condition operator with DeepONet.

2.1. Operator learning

For an operator \mathcal{G} , \mathcal{G} : $u(x) \mapsto f(x)$, the data feed into the DeepONet is in the form $(u, y, \mathcal{G}(u)(y))$. It is obtained by the given initial conditions. The branch network takes $[u(x_1), u(x_2), \cdots, u(x_m)]^T$ as the input, which is the numerical representation of u, and $[b_1, b_2, \cdots, b_p]^T \in \mathbb{R}^{p \times l}$, where $b_k \in \mathbb{R}^l$ for $k = 1, 2, \cdots, p$, as outputs. The trunk network takes y as the input and $[g_1, g_2, \cdots, g_p] \in \mathbb{R}^p$ as outputs. The Unstacked DeepONet net uses FNN as the baseline model and concatenate the function value at sensor locations and the evaluated point together, i.e. $[u(x_1), u(x_2), \cdots, u(x_m), y]^T$. As the equation in the Universal Approximation Theorem for Operators, we can take the product of h and t, then we obtain:

$$\mathcal{G}(\boldsymbol{u})(\boldsymbol{x}) \approx \sum_{k=1}^{p} g_k \boldsymbol{b}_k \tag{6}$$

The activation functions are applied to the trunk net in the last layer. There is no bias in this network. However, according to the theorem 1, the generalization error can be reduced by adding bias. We also give the form with bias b_0 :

$$\mathcal{G}(\boldsymbol{u})(\boldsymbol{x}) \approx \sum_{k=1}^{p} g_k \boldsymbol{b}_k + \boldsymbol{b}_0$$
(7)

As mentioned before, we assumed the initial condition operator has been trained very well. We are going to find the update rule of the parameters to evolve the neural network.

2.2. The evolution of parameters in the neural network

Denoting the parameters in the branch network as W_1 and the parameters in the trunk network as W_2 , W_1 and W_2 can be regarded a function of t since they change in every time step. According to the derivative's chain rule, we have

$$\frac{\partial \boldsymbol{u}}{\partial t} = \frac{\partial \boldsymbol{u}}{\partial W_1} \frac{\partial W_1}{\partial t} + \frac{\partial \boldsymbol{u}}{\partial W_2} \frac{\partial W_2}{\partial t}$$
(8)

Since $\boldsymbol{u} = \sum_{k=1}^{p} g_k \boldsymbol{b}_k = \sum_{k=1}^{p} g_k(W_1(t)) \boldsymbol{b}_k(W_2(t))$, then

$$\frac{\partial \boldsymbol{u}}{\partial t} = \sum_{k=1}^{p} \frac{\partial g_k(W_1(t))}{\partial W_1} \frac{\partial W_1}{\partial t} \boldsymbol{b}_k(W_2(t)) + \sum_{k=1}^{p} g_k(W_1(t)) \frac{\partial \boldsymbol{b}_k(W_2(t))}{\partial W_2} \frac{\partial W_2}{\partial t}$$
(9)

Our objective is to obtain $\frac{\partial W_1}{\partial t}$ and $\frac{\partial W_2}{\partial t}$, the update rule for parameters. It is equivalent to solve a minimization problem,

$$\left[\frac{\partial W_1}{\partial t}; \frac{\partial W_2}{\partial t}\right] = \operatorname{argmin} \mathcal{J}(\gamma_1, \gamma_2)$$
(10)

where

$$\mathcal{J}(\gamma_1, \gamma_2) = \frac{1}{2} \left\| \sum_{k=1}^p \frac{\partial g_k(W_1(t))}{\partial W_1} \gamma_1 \boldsymbol{b}_k(W_2(t)) + \sum_{k=1}^p g_k(W_1(t)) \frac{\partial \boldsymbol{b}_k(W_2(t))}{\partial W_2} \gamma_2 - \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}) \right\|_2^2$$
(11)

In this article, the inner product (a, b) is defined in the integral sense, $(a, b) = \int_{\Omega} a(\mathbf{x})b(\mathbf{x}) d\mathbf{x}$ and the L_2 norm is defined as $||a||_2^2 = \int_{\Omega} |a(\mathbf{x})|^2 d\mathbf{x}$.

The minimization problem can be transformed into a linear system by the first-order optimal condition:

$$\frac{\partial \mathcal{J}}{\partial \gamma_1} = \int_{\Omega} \left(\sum_{k=1}^p \frac{\partial g_k(W_1(t))}{\partial W_1} \boldsymbol{b}_k(W_2(t)) \right)^T \left(\gamma_1 \sum_{k=1}^p \frac{\partial g_k(W_1(t))}{\partial W_1} \boldsymbol{b}_k(W_2(t)) + \sum_{k=1}^p g_k(W_1(t)) \frac{\partial \boldsymbol{b}_k(W_2(t))}{\partial W_2} \gamma_2 - \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}) \right) \mathrm{d}\boldsymbol{x} = 0$$
(12)

$$\frac{\partial \mathcal{J}}{\partial \gamma_2} = \int_{\Omega} \left(\sum_{k=1}^p g_k(W_1(t)) \frac{\partial \boldsymbol{b}_k(W_2(t))}{\partial W_2} \right)^T \left(\gamma_1 \sum_{k=1}^p \frac{\partial g_k(W_1(t))}{\partial W_1} \boldsymbol{b}_k(W_2(t)) + \sum_{k=1}^p g_k(W_1(t)) \frac{\partial \boldsymbol{b}_k(W_2(t))}{\partial W_2} \gamma_2 - \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}) \right) \mathrm{d}\boldsymbol{x} = 0$$
(13)

In this system, the gradient with respect to $W_1(t)$ and $W_2(t)$ can be computed by automatic differentiation at each time step. By denoting

$$(\mathbf{J}_{1})_{ij_{1}} = \sum_{k=1}^{p} \frac{\partial g_{k}(W_{1}(t))}{\partial W_{1}^{j_{1}}} \boldsymbol{b}_{k}^{i}(W_{2}(t))$$
(14)

$$(\mathbf{J}_{2})_{ij_{2}} = \sum_{k=1}^{p} g_{k}(W_{1}(t)) \frac{\partial \boldsymbol{b}_{k}^{i}(W_{2}(t))}{\partial W_{2}^{j_{2}}}$$
(15)

$$(\mathbf{N})_i = \mathcal{N}\left(\boldsymbol{u}_x^i\right) \tag{16}$$

where $i = 1, 2, \dots, l$, $j_1 = 1, 2, \dots, N_{\text{para}}^b$, $j_2 = 1, 2, \dots, N_{\text{para}}^t$. N_{para}^b is the number of parameters in Branch net and N_{para}^t is the number of parameters in Trunk net. **N** is generated by the DeepONet, so it can be evaluated at any spatial point. The above integrals can be approximated by numerical methods:

$$\frac{1}{|\Omega|} \int_{\Omega} \left(\sum_{k=1}^{p} \frac{\partial g_k(W_1(t))}{\partial W_1} \boldsymbol{b}_k(W_2(t)) \right)^T \left(\sum_{k=1}^{p} \frac{\partial g_k(W_1(t))}{\partial W_1} \boldsymbol{b}_k(W_2(t)) \right) d\boldsymbol{x} = \lim_{l \to \infty} \frac{1}{l} \mathbf{J}_1^{\mathrm{T}} \mathbf{J}_1$$
(17)

$$\frac{1}{\Omega|} \int_{\Omega} \left(\sum_{k=1}^{p} g_k(W_1(t)) \frac{\partial \boldsymbol{b}_k(W_2(t))}{\partial W_2} \right)^T \left(\sum_{k=1}^{p} g_k(W_1(t)) \frac{\partial \boldsymbol{b}_k(W_2(t))}{\partial W_2} \right) \mathrm{d}\boldsymbol{x} = \lim_{l \to \infty} \frac{1}{l} \mathbf{J}_2^{\mathbf{T}} \mathbf{J}_2$$
(18)

$$\frac{1}{|\Omega|} \int_{\Omega} \left(\sum_{k=1}^{p} \frac{\partial g_k(W_1(t))}{\partial W_1} \boldsymbol{b}_k(W_2(t)) \right)^T (\mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u})) \, \mathrm{d}\boldsymbol{x} = \lim_{l \to \infty} \frac{1}{l} \mathbf{J}_1^{\mathrm{T}} \mathbf{N}$$
(19)

By denoting γ_i^{opt} as optimal values of γ_i , i = 1, 2, the objective function can be reduced to

$$\mathbf{J}_{\mathbf{1}}^{\mathrm{T}}\left(\boldsymbol{\gamma}_{1}^{opt}\mathbf{J}_{\mathbf{1}}+\boldsymbol{\gamma}_{2}^{opt}\mathbf{J}_{\mathbf{2}}-\mathbf{N}\right)=0$$
(20)

$$\mathbf{J}_{\mathbf{2}}^{T}\left(\boldsymbol{\gamma}_{1}^{opt}\mathbf{J}_{\mathbf{1}}+\boldsymbol{\gamma}_{2}^{opt}\mathbf{J}_{\mathbf{2}}-\mathbf{N}\right)=0$$
(21)

The feasible solutions of the above equations are the approximated time derivatives of W_1 and W_2 .

$$\frac{dW_1}{dt} = \gamma_1^{opt} \tag{22}$$

$$\frac{dW_2}{dt} = \gamma_2^{opt} \tag{23}$$

where the initial conditions W_1^0 and W_2^0 can be determined by DeepONets for initial condition operators. The two ODEs are the updated rules in the neural networks. The simple way to solve them is the explicit Euler method.

$$\frac{W_1^{n+1} - W_1^n}{\Delta t} = \gamma_1^{opt} \tag{24}$$

$$\frac{W_2^{n+1} - W_2^n}{\Delta t} = \gamma_2^{opt} \tag{25}$$

The neural network can calculate the solution of given PDEs at any time step t_n and spatial point x_i by weights W_1^n , W_2^n , spatial points x and initial condition u(x).

3. Energy Dissipative Evolutionary Deep Operator Neural Network

Let's reconsider the given problem.

$$\frac{\partial u}{\partial t} + \mathcal{N}_{\mathbf{x}}(u) = 0 \tag{26}$$
$$u(\mathbf{x}, 0) = f(\mathbf{x})$$

where $u \in \mathbb{R}^l$, $\mathcal{N}_x(u)$ can be written as a variational derivative of a free energy functional E[u(x)] bounded from below, $\mathcal{N}_x(u) = \frac{\delta E}{\delta u}$. Taking the inner product with $\mathcal{N}_x(u)$ of the first equation, we obtain the energy dissipation property

$$\frac{dE[\boldsymbol{u}(\boldsymbol{x})]}{dt} = \left(\frac{\delta E}{\delta \boldsymbol{u}}, \frac{\partial \boldsymbol{u}}{\partial t}\right) = \left(\mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}), \frac{\partial \boldsymbol{u}}{\partial t}\right) = -\left(\mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}), \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u})\right) \le 0$$
(27)

However, it is usually hard for a numerical algorithm to be efficient as well as energy dissipative. Recently, the SAV approach [33] was introduced to construct numerical schemes which is energy dissipative (with a modified energy), accurate, robust and easy to implement. More precisely, assuming E[u(x)] > 0, it introduces a $r(t) = \sqrt{E[u(x,t)]}$, and expands the gradient flow problem as

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\frac{r}{\sqrt{E(\boldsymbol{u})}} \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u})$$

$$r_t = \frac{1}{2\sqrt{E(\boldsymbol{u})}} \left(\mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}), \frac{\partial \boldsymbol{u}}{\partial t} \right)$$
(28)

With $r(0) = \sqrt{E[u(x, t)]}$, the above system has a solution $r(t) \equiv \sqrt{E[u(x, t)]}$ and u being the solution of the original problem.

3.1. First order scheme

By setting $u^n = \sum_{k=1}^p g_k b_k$, a first order scheme can be constructed as

(r

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} = -\frac{r^{n+1}}{\sqrt{E(\boldsymbol{u}^n)}} \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}^n)$$

$$\frac{r^{n+1} - r^n}{\Delta t} = \frac{1}{2\sqrt{E(\boldsymbol{u}^n)}} \int_{\Omega} \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}^n) \frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} d\boldsymbol{x}.$$
(29)

This is a coupled system of equations for (r^{n+1}, u^{n+1}) . But it can be easily decoupled as follows. Plugging the first equation into the second one, we obtain:

$$\frac{r^{n+1} - r^n}{\Delta t} = -\frac{r^{n+1}}{2E(\boldsymbol{u}^n)} \|\mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}^n)\|^2,$$
(30)

which implies

$$r^{n+1} = \left(1 + \frac{\Delta t}{2E(u^n)} \left\|\mathcal{N}_x(u^n)\right\|^2\right)^{-1} r^n$$
(31)

Theorem 3.1 (Discrete Energy Dissipation Law). With the modified energy define above, the scheme is unconditionally energy stable, i.e.

$$(r^{n+1})^2 - (r^n)^2 \le 0.$$
(32)

Proof 3.1. Taking the inner product of the first equation with $\frac{r^{n+1}}{\sqrt{E(u^n)}} \mathcal{N}_x(u^n)$ and the second equation with $2r^{n+1}$

$${}^{n+1})^{2} - (r^{n})^{2} = 2r^{n+1}(r^{n+1} - r^{n}) - (r^{n+1} - r^{n})^{2}$$

$$= \frac{\Delta tr^{n+1}}{\sqrt{E(u^{n})}} \int_{\Omega} \mathcal{N}_{x}(u^{n}) \frac{u^{n+1} - u^{n}}{\Delta t} dx - (r^{n+1} - r^{n})^{2}$$

$$= -\left(\frac{r^{n+1}}{\sqrt{E(u^{n})}}\right)^{2} \int_{\Omega} \mathcal{N}_{x}(u^{n}) \mathcal{N}_{x}(u^{n}) dx - (r^{n+1} - r^{n})^{2}$$

$$\leq 0$$
(33)

In order to maintain the modified energy dissipation law in the evolution neural network, we only need to replace $\mathcal{N}_x(u)$ by $\frac{t^{n+1}}{\sqrt{\mathcal{F}(u)}}\mathcal{N}_x(u)$ in section 2. The update rule of the neural network is

$$\left[\frac{\partial W_1}{\partial t}; \frac{\partial W_2}{\partial t}\right] = \operatorname{argmin} \mathcal{J}(\gamma_1, \gamma_2)$$
(34)

where

$$\mathcal{J}(\gamma_1, \gamma_2) = \frac{1}{2} \left\| \sum_{k=1}^p \frac{\partial g_k(W_1^n)}{\partial W_1^n} \gamma_1 \boldsymbol{b}_k(W_2^n) + \sum_{k=1}^p g_k(W_1^n) \frac{\partial \boldsymbol{b}_k(W_2^n)}{\partial W_2^n} \gamma_2 - \frac{r^{n+1}}{\sqrt{E(\boldsymbol{u}^n)}} \mathcal{N}_{\boldsymbol{x}}(\boldsymbol{u}^n) \right\|_2^2$$
(35)

The corresponding linear system of the first order optimal condition is

$$\mathbf{J}_{\mathbf{1}}^{\mathbf{T}}\left(\gamma_{1}^{opt}\mathbf{J}_{\mathbf{1}}+\gamma_{2}^{opt}\mathbf{J}_{\mathbf{2}}-\frac{r^{n+1}}{\sqrt{E(\boldsymbol{u}^{n})}}\mathbf{N}\right)=0$$
(36)

$$\mathbf{J}_{2}^{\mathbf{T}}\left(\gamma_{1}^{opt}\mathbf{J}_{1}+\gamma_{2}^{opt}\mathbf{J}_{2}-\frac{r^{n+1}}{\sqrt{E(\boldsymbol{u}^{n})}}\mathbf{N}\right)=0$$
(37)

where

$$(\mathbf{J}_1)_{ij_1} = \sum_{k=1}^p \frac{\partial g_k(W_1^n)}{\partial W_1^{n,j_1}} \boldsymbol{b}_k^i(W_2^n)$$
(38)

$$(\mathbf{J}_{2})_{ij_{2}} = \sum_{k=1}^{p} g_{k}(W_{1}^{n}) \frac{\partial \boldsymbol{b}_{k}^{i}(W_{2}^{n})}{\partial W_{2}^{n,j_{2}}}$$
(39)

$$(\mathbf{N})_i = \mathcal{N}\left(\boldsymbol{u}_x^i\right) \tag{40}$$

and $i = 1, 2, \dots, l$, $j_1 = 1, 2, \dots, N_{\text{para}}^b$, $j_2 = 1, 2, \dots, N_{\text{para}}^t$. N_{para}^b is the number of parameters in Branch net and N_{para}^t is the number of parameters in Trunk net. After getting γ_1^{opt} and γ_2^{opt} , W^{n+1} can be obtained by the Forward Euler method as equation (24) and (25).

$$W_1^{n+1} = W_1^n + \gamma_1^{opt} \Delta t$$
(41)

$$W_2^{n+1} = W_2^n + \gamma_2^{opt} \Delta t$$
 (42)

4. Adaptive time stepping strategy and Restart strategy

One of the advantages of an unconditionally stable scheme is that the adaptive time step can be utilized. Since the coefficient of N_x , $\frac{t^{n+1}}{\sqrt{E^n}}$ should be around 1, by denoting $\xi^{n+1} = \frac{t^{n+1}}{\sqrt{E^n}}$, larger Δt is allowed when ξ is close to 1 and the smaller Δt is needed when ξ is far away from 1. Thus, a simple adaptive time-stepping strategy can be described as follows:

Algorithm 1 Adaptive time stepping strategy

1. Set the tolerance for ξ as ϵ_0 and ϵ_1 , the initial time step Δt , the maximum time step Δt_{max} and the minimum time step Δt_{min} 2. Compute u^{n+1} .

3. Compute $t^{n+1} = \frac{t^{n+1}}{\sqrt{E^n}}$. 4. If $|1 - \xi^{n+1}| > \epsilon_0$, Then $\Delta t = \max(\Delta t_{min}, \Delta t/2)$; Else if $|1 - \xi^{n+1}| < \epsilon_1$, Then $\Delta t = \min(\Delta t_{max}, 2\Delta t)$. Go to Step 2. 5. Update time step Δt .

Another popular strategy to keep r approximating the original energy E is to reset the SAV r^{n+1} to be E^{n+1} in some scenarios. The specific algorithm is as following:

Algorithm 2 Restart strategy

 Set the tolerance for ξ as ε₂.
 Compute uⁿ⁺¹.
 Compute ξⁿ⁺¹ = rⁿ⁺¹/√Eⁿ.
 If |1 - ξⁿ⁺¹| > ε₂, Then rⁿ⁺¹ = √Eⁿ⁺¹ and Go to Step 2.
 Go to next iteration.

The choice for ϵ_0 , ϵ_1 should be some small tolerance, usually 10^{-1} and 10^{-3} . The choices for Δt_{max} and Δt_{min} are quite dependent on Δt , usually $\Delta t_{max} = 10^3 \times \Delta t$ and $\Delta t_{min} = 10^{-3} \times \Delta t$. In Algorithm 2, we usually take ϵ_2 as 2×10^{-2} .

5. Algorithm for EDE-DeepONet

A general approach to solving a time-dependent PDE with EDE-DeepONet can be summarized in Algorithm 3.

Algorithm 3 Energy-Dissipative Evolutionary Deep Operator Neural Networks(EDE-DeepONet)

1. Generate input data samples in the form of $(u, y, \mathcal{G}(u)(y))$ for the DeepONet, where \mathcal{G} is the objective operator. Each specific input function *u* can be generated in the same sensor locations $\{x_1, x_2, \dots, x_m\}$.

2. Feed $[u(x_1), u(x_2), \dots, u(x_m)]$ into the branch network and $y \in Y$ into the trunk network. Denote the output of the DeepONet as q.

3. Update the parameters in the DeepONet by minimizing a cost function, where the cost function can be taken as the mean squared error as $\frac{1}{|Y|} \sum_{y \in Y} ||\mathcal{G}(u)(y) - q||^2$.

4. Once the DeepONet has been trained well, solve the system of equations of (36) and (37) to obtain $\left[\frac{\partial W_1}{\partial t}; \frac{\partial W_2}{\partial t}\right]$.

5. The value of $\left[\frac{\partial W_1}{\partial t}; \frac{\partial W_2}{\partial t}\right]$ can be obtained in the current step. Since the parameters W_1^n in the branch network, and W_2^n in the trunk network are known, W_1^{n+1} and W_2^{n+1} for the next step can be also obtained by the Forward Euler method or Runge-Kutta method.

6. Repeat step 5 until the final time T, where $T = t_0 + s\Delta t$, t_0 is the initial time of the given PDE, Δt is the time step in step 5 and s is the number of repeated times of step 5.

7. Output the solution at time *T* in the DeepONet with parameters obtained in step 6.

6. Numerical Experiments

In this section, we implement EDE-DeepONet to solve heat equations, parametric heat equations, and Allen-Cahn equations to show its performance and accuracy.

6.1. Example 1: Simple heat equations

To show the accuracy of the EDE-DeepONet, we start with the simple heat equation with different initial conditions since we already have the exact solution. A 1D heat equation system can be described by

$$u_t = u_{xx} \tag{43}$$

$$u(x,0) = f \tag{44}$$

$$u(0,t) = u(2,t) = 0 \tag{45}$$

By the method of separation of variables, we can derive the solution to the heat equation. If we set $f(x) = asin(\pi x)$, the solution is $u(x,t) = asin(\pi x)e^{-\pi^2 t}$, where $a \in [1,2]$. The corresponding energy is $E(u) = \int_0^2 \frac{1}{2}|u_x|^2 dx \approx \Delta x(\sum_{i=1}^n \frac{1}{2}|u_x(x_i)|^2)$. With different parameters *a*, the above equation describes a kind of PDE. The input data samples can be generated as $(a, x, \mathcal{G}(a)(x))$, where $\mathcal{G}(a)(x) = a \sin(\pi x)$ for specific *a* and *x*. When generating the initial data samples, we choose 50 points from [0, 2) uniformly for x and 50 random values of *a* from [1, 2]. The time step when updating the parameters in the neural network is 2.5×10^4 . The number of iteration steps is 400. We compared the different solutions with 4 different *a*, 1.0, 1.5, 1.8, 2.5 every 100 steps. Although a = 2.5 is out of the range of training

data, it still performs well in this model. With the exact solution, we also get the error with different *a* as Table 1. The error is defined by $\frac{1}{N_x} \sum_{k=1}^{N_x} (u(x_k) - \hat{u}(x_k))^2$, where $N_x = 51$, *u* is the solution obtained by EDE-DeepONet and \hat{u} is the exact solution. To illustrate the relationship between the modified energy and the original energy, we compare r^2 and *E* at each step as Figure 2. Both energy are actually disspative in the EDE-DeepONet except when restart strategy applied. The restart strategy is used to keep r^2 approaching *E*. The modified energy is initialized when the restart strategy applied. The restart strategy was triggered on the 370th step since the modified energy approaches the original energy are offset. After that, they are on the same trajectory again. It is clear that the modified energy approaches the original energy before and after the restart strategy applied. In Figure 3, we give the comparison between the exact solution and the solution obtained by EDE-DeepONet. From this simple heat equation, we show that EDE-DeepONet correctly predicts the solution of the PDE. The most important fact is that EDE-DeepONet can not only predict the solution in the training subset range but also the solution out of the training range. For instance, we take a = 2.5 while $a \in [1, 2]$ in the training process. EDE-DeepONet shows good accuracy compared to the exact solution as Figure 3 (a)-(d) and Table 1.



Fig. 2: The heat equation: The modified energy and original energy when training the network. Each iteration step represents one forward step of the PDE's numerical solution with $\Delta t = 2.5 \times 10^{-4}$. In the EDE-DeepONet, both energy are actually dissipative, with the exception of the restart strategy. In order to maintain that r^2 approaches *E*, the restart strategy is employed, and the modified energy is initialized on the 370th step. The modified energy and the original energy on the same trajectory before and after the 370th step.

Error	T = 0.025	T = 0.05	T = 0.075	T = 0.1
<i>a</i> = 1.0	1.47×10^{-5}	1.33×10^{-5}	1.32×10^{-5}	1.29×10^{-5}
<i>a</i> = 1.5	5.11×10^{-6}	7.05×10^{-6}	8.48×10^{-6}	9.81×10^{-6}
<i>a</i> = 1.8	1.46×10^{-5}	1.69×10^{-5}	1.79×10^{-5}	1.83×10^{-5}
<i>a</i> = 2.5	2.20×10^{-4}	1.34×10^{-4}	6.02×10^{-5}	1.72×10^{-5}

Table 1: The heat equation: The initial condition of the PDE is $f(x) = a \sin(\pi x)$. The error is defined by $\frac{1}{N_x} \sum_{k=1}^{N_x} (u(x_k) - \hat{u}(x_k))^2$, where $N_x = 51$, *u* is the solution obtained by EDE-DeepONet and \hat{u} is the exact solution.

6.2. Example 2: Parametric heat equations

In example 1, we take different initial conditions as our inputs. In example 2, we are going to deal with the parametric heat equations. A general parametric heat equation in 1D can be described by



Fig. 3: The heat equation: The solution with 4 different initial conditions $f(x) = a \sin(\pi x)$. The curve represents the solution obtained by the EDE-DeepONet, and xxx represents the reference solution. The training parameter *a* is in the range of [1, 2), so we give three examples in this range. We also present the case out of the range. It also shows accuracy in Figure 3-(d).

$$u_t = c u_{xx} \tag{46}$$

$$u(x,0) = \sin(\pi x) \tag{47}$$

$$u(0,t) = u(2,t) = 0 \tag{48}$$

This PDE is more complex than the PDE in Example 1 since the parameter is inside the equation. The traditional numerical scheme needs to be run multiple times to deal with the case with different parameters because they are actually different equations. However, we only need to train the EDE-DeepONet once. The training range of c is chosen as [1, 2). We choose 50 points of x and c in the same way as example 1. First, we compared the modified energy with the original energy as Figure 4. The energy is not the same as the first example since the energy depends on the parameter c. We compute the average of the energy with different c to represent the energy of the system. This case is more complex than the first one, so it needs more restarts during the training. Even though the modified energy oscillates when restart strategy used, it keeps decreasing after each restart. Second, we give the error between the solution obtained by the EDE-DeepONet and the reference solution in Table 2, where the reference solution can be obtained explicitly by variable separation method and the error is defined in the same way as example 1. Third, we give the comparison between our solution and the reference solution in Figure 5. Same as example 1, we give the predicted solution of $c \notin [1, 2]$. All of them show the good accuracy. Hence, EDE-DeepONet can actually solve parametric PDEs.

6.3. Example 3: Allen-Cahn equations

The energy in Examples 1 and 2 is quadratic and the right-hand side of the PDE is linear with respect to u. We are going to show the result for the PDE with more complicated energy. The Allen-Cahn equation is a kind of reaction-diffusion equation. It is derived to describe the process of the phase separation. It was developed to solve a problem in the material science area and has been used to represent the moving interfaces in a phase-field model



Fig. 4: The parametric heat equation: The modified energy and original energy when training the network. Each iteration step represents one forward step of the PDE's numerical solution with $\Delta t = 2.5 \times 10^{-4}$. This kind of PDEs is more complicated, so it need more restarts in the training process. The original energy keeps decreasing and the modified energy also shows good approximation of the original energy.

Error	T = 0.025	T = 0.05	T = 0.075	T = 0.1
<i>c</i> = 1.2	1.30×10^{-5}	1.43×10^{-5}	1.35×10^{-5}	1.20×10^{-5}
<i>c</i> = 1.5	1.35×10^{-5}	1.27×10^{-5}	9.80×10^{-6}	7.80×10^{-6}
<i>c</i> = 1.8	1.17×10^{-5}	1.03×10^{-5}	7.88×10^{-5}	1.83×10^{-5}
<i>c</i> = 2.5	2.20×10^{-4}	1.34×10^{-4}	6.02×10^{-5}	7.08×10^{-6}

Table 2: The parametric heat equation: The initial condition of the PDE is $f(x) = \sin(\pi x)$. The error is defined by $\frac{1}{N_x} \sum_{k=1}^{N_x} (u(x_k) - \hat{u}(x_k))^2$, where $N_x = 51$, *u* is the solution obtained by EDE-DeepONet and \hat{u} is the exact solution.

in fluid dynamics. The Allen-Cahn equation can be treated as a gradient flow in L^2 with some specific energy. We discussed the 1D case and 2D case as follows:

6.3.1. 1D case

(a) Various initial conditions:

We start with the simple case, 1D Allen-Cahn equation. It can be described by the following equations:

$$u_t = u_{xx} - g(x) \tag{49}$$

$$u(x,0) = a\sin\pi x \tag{50}$$

$$u(-1,t) = u(1,t) = 0 \tag{51}$$

The corresponding Ginzburg–Landau free energy $E[u] = \int_0^1 \frac{1}{2} |u_x|^2 dx + \int_{x=0}^{x=1} G(u) dx$, where $G(u) = \frac{1}{4\epsilon^2}(u^2 - 1)^2$ and $g(u) = G'(u) = \frac{1}{\epsilon^2}u(u^2 - 1)$, $\epsilon = 0.1$. The parameter ϵ affects the width of the jump when arriving at the steady state as the Figure 7 (c), (j), (o) and (t). In the EDE-DeepONet, we set $\Delta t = 10^{-4}$, the number of spatial points N_x is 51 and the range of *a* is [0.1, 0.5]. We also compared the modified energy and the original energy as Figure 6. The modified energy can approximate well to the original energy even in a much more complicated form. Then, we compared 4 different solutions with different $a \in [0.1, 0.5]$ obtained by the EDE-DeepONet and the reference solution obtained by the SAV method in traditional numerical computation as Figure 7. The error is shown in Table 3, where error is



Fig. 5: The parametric heat equation: The solution with 4 different parameters c. The curve represents the solution obtained by the EDE-DeepONet and xxx represents the reference solution. The training parameter c is in the range of [1, 2), so we give 3 examples in this range. We also present the case out of the range in Figure 5-(d).

defined in the same way as example 1. $a = 0.6 \notin [0.1, 0.5)$ shows that EDE-DeepONet can predict the solution well out of the training range. We compared the solution with 4 different initial condition parameter *a* every 100 steps until the final time T = 0.04 as Figure 7. Each row presents the solution under the same initial condition but with different evolution time *T*. With this example, it shows that EDE-DeepONet can deal with the PDE with a jump, while it is hard for other neural networks.

Error	T = 0.01	T = 0.02	T = 0.03	T = 0.04
a = 0.1	4.95×10^{-5}	2.27×10^{-4}	4.97×10^{-4}	7.40×10^{-4}
a = 0.3	2.32×10^{-4}	4.62×10^{-4}	6.66×10^{-4}	7.58×10^{-4}
a = 0.2	1.25×10^{-4}	3.46×10^{-4}	4.86×10^{-3}	7.15×10^{-4}
<i>a</i> = 0.4	2.80×10^{-4}	4.83×10^{-4}	6.35×10^{-4}	7.09×10^{-4}
<i>a</i> = 0.6	6.45×10^{-4}	5.02×10^{-4}	4.90×10^{-4}	6.15×10^{-4}

Table 3: 1D Allen-Cahn equation: The initial condition of the Allen-Cahn equation is $f(x) = a \sin(\pi x)$. The error is defined by $\frac{1}{N_x} \sum_{k=1}^{N_x} (u(x_k) - \hat{u}(x_k))^2$, where $N_x = 51$, *u* is the solution obtained by EDE-DeepONet and \hat{u} is the reference solution.

(b) Various thickness of the interface:

Heuristically, ϵ represents the thickness of the interface in the phase separation process. We are able to obtain a sharp interface when $\epsilon \rightarrow 0$ with evolving in time. Each theoretical and numerical analysis of the limit makes a difference in the purpose of the understanding of the equation, cf. e.g. [34, 35]. We take ϵ as a training parameter. The problem can be described as:

$$u_t = u_{xx} - \frac{1}{\epsilon^2} (u^3 - u) \tag{52}$$

$$u(-1,t) = u(1,t) = 0 \tag{53}$$



Fig. 6: 1D Allen-Cahn equation: The modified energy and original energy when training the network are shown above. Each iteration step represents one forward step of the PDE's numerical solution with $\Delta t = 10^{-4}$. The modified energy shows the same trends as the original energy.

Since the training sample contains the parameter ϵ , we can not use the same initial condition as the last example. We use spectral methods for a few steps with initial condition $u(x, 0) = 0.4 \sin(\pi x)$. The training sample is generated based on the numerical solution of $u_{\epsilon}(x, 0.02)$. We randomly select 50 different ϵ from [0.1, 0.2]. We set the learning rate as $\Delta t = 10^{-4}$ and apply the adaptive time stepping strategy. We obtain the predicted solution after 400 iterations with different ϵ . The rest setting is the same as the last example. The solution with different ϵ is shown in Figure 8. As ϵ goes smaller, the interface is sharper. Besides, the range of the training parameter is (0.1, 0.2). We are also able to obtain the solution out of the above range. EDE-DeepONet can track the limit of ϵ in only one training process, whereas other traditional numerical methods hardly make it.

6.3.2. 2D case

The 2D case Allen-Cahn equation is even more complex. The problem can be described as follows:

$$u_t = \Delta u - g(u) \tag{54}$$

$$u(x, y, 0) = a\sin(\pi x)\sin(\pi y)$$
(55)

$$u(-1, y, t) = u(1, y, t) = u(x, -1, t) = u(x, 1, t) = 0$$
(56)

The corresponding Ginzburg–Landau free energy $E[u] = \int_{-1}^{1} \int_{-1}^{1} \frac{1}{2} (|u_x|^2 + |u_y|^2) dx dy + \int_{-1}^{1} \int_{-1}^{1} G(u) dx$, where $G(u) = \frac{1}{4\epsilon^2} (u^2 - 1)^2$ and $g(u) = G'(u) = \frac{1}{\epsilon^2} u(u^2 - 1)$. Usually, we take $\epsilon = 0.1$. In the training process, we take $\Delta t = 2 \times 10^{-4}$. The number of spatial points is 51×51 and the number of training parameters *a* is 20. The way to choose $a \in (0.1, 0.4)$ and *x* is the same as in example 1. We first compared the exact solution and the solution obtained by EDE-DeepONet with initial condition $f(x, y) = 0.2 \sin(\pi x) \sin(\pi y)$, where the exact solution is obtained by the traditional SAV method. EDE-DeepONet predicts the solution correctly based on Table 4 and Figure 9. Then in order to show its accuracy, we draw Figure 10 with more parameters. All the examples show good trends to separate. The case a = 0.4 is out of the training range, but it still approaches the exact solution.



Fig. 7: 1d Allen-Cahn equation: The solution for 1d Allen-Cahn equation with 4 different initial conditions $f(x) = a \sin \pi x$. The curve represents the solution obtained by our model, and xxx represents the reference solution. We draw the figure for every 100 steps. The range of *a* is [0.1, 0.5]. We also compare the solution with $a \notin [0.1, 0.5]$. All the figures show the trends of the phase separation.

Error	T = 0.01	T = 0.02	T = 0.03
<i>a</i> = 0.15	1.23×10^{-4}	6.53×10^{-4}	2.75×10^{-3}
a = 0.2	2.24×10^{-4}	1.10×10^{-3}	4.04×10^{-3}
a = 0.3	4.28×10^{-4}	1.84×10^{-3}	5.76×10^{-3}
a = 0.35	5.31×10^{-4}	2.17×10^{-3}	6.25×10^{-3}
<i>a</i> = 0.4	6.94×10^{-4}	2.71×10^{-3}	7.22×10^{-3}

Table 4: 2D Allen-Cahn equation: The initial condition of the 2D Allen-Cahn equation is $f(x, y) = a \sin(\pi x) \sin(\pi y)$. The error is defined by $\frac{1}{N_x N_y} \sum_{k=1}^{N_x} \sum_{i=1}^{N_y} (u(x_k, y_i) - \hat{u}(x_k, y_i))^2$, where $N_x = N_y = 51$, *u* is the solution obtained by EDE-DeepONet and \hat{u} is the reference solution.

7. Concluding Remarks

In this paper, we provide a new neural network architecture to solve parametric PDEs with different initial conditions, while maintaining the energy dissipative of dynamic systems. We first introduce the energy dissipative law of dynamic systems to the DeepONet. We also introduce an adaptive time stepping strategy and restart strategy. With our experiments, both above strategies help keep the modified energy approaching the original energy. To avoid much cost of training the DeepONet, we evolve the neural network based on Euler methods. In this article, we adopt the SAV method to solve gradient flow problems. With this successful attempt, more work could be done. For example, we can consider a general Wasserstein gradient flow problem. We are only adopting the basic architecture of the DeepONet. The more advanced architecture is compatible to our work. It may further improve the accuracy of EDE-DeepONet.

Acknowledgments

SJ and SZ gratefully acknowledge the support of NSF DMS-1720442 and AFOSR FA9550-20-1-0309. GL and ZZ gratefully acknowledge the support of the National Science Foundation (DMS-1555072, DMS-2053746, and DMS-



Fig. 8: 1d Allen-Cahn equation: Solutions with different thickness of the interface at the same final time. The curve represents the solution obtained by EDE-DeepONet. xxx represents the reference solution.

2134209), Brookhaven National Laboratory Subcontract 382247, and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program DE-SC0021142 and DE-SC0023161.



Fig. 9: 2D Allen-Cahn equation: (a)-(d) represents the reference solution of the 2D Allen-Cahn equation with initial condition $f(x, y) = 0.3 \sin(\pi x) \sin(\pi y)$. (e)-(h) is the solution obtained by the EDE-DeepONet.



Fig. 10: 2D Allen-Cahn equation: The solution of 2D Allen-Cahn equation with 4 different initial conditions $f(x, y) = a \sin \pi x \sin \pi y$. The training parameter $a \in [0.1, 0.4]$. We draw three figures where a is in the training range and one figure where a is out of the training range. All the figures show the phase separation trends according to the reference solution. As a is further away from the training range, the error tends to be larger.

References

- N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces, arXiv preprint arXiv:2108.08481 (2021).
- [2] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, arXiv preprint arXiv:2003.03485 (2020).
- [3] Y. Khoo, J. Lu, L. Ying, Solving parametric pde problems with artificial neural networks, European Journal of Applied Mathematics 32 (2021) 421–435.
- [4] K. Bhattacharya, B. Hosseini, N. B. Kovachki, A. M. Stuart, Model reduction and neural networks for parametric pdes, arXiv preprint arXiv:2005.03180 (2020).
- [5] N. H. Nelsen, A. M. Stuart, The random feature model for input-output maps between banach spaces, SIAM Journal on Scientific Computing 43 (2021) A3212–A3243.
- [6] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).
- [7] R. G. Patel, N. A. Trask, M. A. Wood, E. C. Cyr, A physics-informed operator regression framework for extracting data-driven continuum models, Computer Methods in Applied Mechanics and Engineering 373 (2021) 113500.
- [8] J. A. Opschoor, C. Schwab, J. Zech, Deep learning in high dimension: Relu network expression rates for bayesian pde inversion, SAM Research Report 2020 (2020).
- [9] C. Schwab, J. Zech, Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in uq, Analysis and Applications 17 (2019) 19–55.
- [10] T. O'Leary-Roseberry, U. Villa, P. Chen, O. Ghattas, Derivative-informed projected neural networks for high-dimensional parametric maps governed by pdes, Computer Methods in Applied Mechanics and Engineering 388 (2022) 114199.
- [11] K. Wu, D. Xiu, Data-driven deep learning of partial differential equations in modal space, Journal of Computational Physics 408 (2020) 109307.
- [12] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (2021) 218–229.
- [13] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (1989) 303-314.
- [14] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural networks 4 (1991) 251–257.
- [15] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (1989) 359–366.
- [16] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Transactions on Neural Networks 6 (1995) 911–917.
- [17] M. Raissi, P. Perdikaris, G. E. Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, arXiv preprint arXiv:1801.01236 (2018).
- [18] T. Qin, Z. Chen, J. D. Jakeman, D. Xiu, Deep learning of parameterized equations with applications to uncertainty quantification, International Journal for Uncertainty Quantification 11 (2021).
- [19] N. Winovich, K. Ramani, G. Lin, Convpde-uq: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains, Journal of Computational Physics 394 (2019) 263–279.
- [20] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, Journal of Computational Physics 394 (2019) 56–81.
- [21] J. del Águila Ferrandis, M. S. Triantafyllou, C. Chryssostomidis, G. E. Karniadakis, Learning functionals via lstm neural networks for predicting vessel dynamics in extreme sea states, Proceedings of the Royal Society A 477 (2021) 20190897.
- [22] J. Jia, A. R. Benson, Neural jump stochastic differential equations, Advances in Neural Information Processing Systems 32 (2019).
- [23] T. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, in 'advances in neural information processing systems', La Jolla (2018).
- [24] Y. Du, T. A. Zaki, Evolutional deep neural network, Physical Review E 104 (2021) 045303.
- [25] S. M. Allen, J. W. Cahn, A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening, Acta metallurgica 27 (1979) 1085–1095.
- [26] D. M. Anderson, G. B. McFadden, A. A. Wheeler, Diffuse-interface methods in fluid mechanics, Annual review of fluid mechanics 30 (1998) 139–165.
- [27] J. W. Cahn, J. E. Hilliard, Free energy of a nonuniform system. i. interfacial free energy, The Journal of chemical physics 28 (1958) 258–267.
- [28] M. Doi, S. F. Edwards, S. F. Edwards, The theory of polymer dynamics, volume 73, oxford university press, 1988.
- [29] K. Elder, M. Katakowski, M. Haataja, M. Grant, Modeling elasticity in crystal growth, Physical review letters 88 (2002) 245701.
- [30] M. E. Gurtin, D. Polignone, J. Vinals, Two-phase binary fluids and immiscible fluids described by an order parameter, Mathematical Models and Methods in Applied Sciences 6 (1996) 815–831.
- [31] F. M. Leslie, Theory of flow phenomena in liquid crystals, in: Advances in liquid crystals, volume 4, Elsevier, 1979, pp. 1-81.
- [32] P. Yue, J. J. Feng, C. Liu, J. Shen, A diffuse-interface method for simulating two-phase flows of complex fluids, Journal of Fluid Mechanics 515 (2004) 293–317.
- [33] J. Shen, J. Xu, J. Yang, The scalar auxiliary variable (sav) approach for gradient flows, Journal of Computational Physics 353 (2018) 407-416.
- [34] G. Caginalp, X. Chen, Convergence of the phase field model to its sharp interface limits, European Journal of Applied Mathematics 9 (1998) 417–445.
- [35] X. Chen, G. Caginalp, C. Eck, A rapidly converging phase field model, Discrete & Continuous Dynamical Systems 15 (2006) 1017.