INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 20, Number 5, Pages 693–708

doi: 10.4208/ijnam2023-1030

NUMERICAL APPROXIMATIONS OF THE ALLEN-CAHN-OHTA-KAWASAKI EQUATION WITH MODIFIED PHYSICS-INFORMED NEURAL NETWORKS (PINNS)

JINGJING XU, JIA ZHAO, AND YANXIANG ZHAO*

Abstract. The physics-informed neural networks (PINNs) has been widely utilized to numerically approximate PDE problems. While PINNs has achieved good results in producing solutions for many partial differential equations, studies have shown that it does not perform well on phase field models. In this paper, we partially address this issue by introducing a modified physics-informed neural networks. In particular, they are used to numerically approximate Allen-Cahn-Ohta-Kawasaki (ACOK) equation with a volume constraint. Technically, the inverse of Laplacian in the ACOK model presents many challenges to the baseline PINNs. To take the zero-mean condition of the inverse of Laplacian, as well as the volume constraint, into consideration, we also introduce a separate neural network, which takes the second set of sampling points in the approximation process. Numerical results are shown to demonstrate the effectiveness of the modified PINNs. An additional benefit of this research is that the modified PINNs can also be applied to learn more general nonlocal phase-field models, even with an unknown nonlocal kernel.

Key words. Physics-informed neural networks, Allen-Cahn-Ohta-Kawasaki equation, phase field models.

1. Introduction

Ohta-Kawasaki(OK) free energy [22], usually associated with a volume constraint, has been used to simulate the phase separation of diblock copolymers, i.e., polymers consisting of two types of monomers, A and B, that are chemically incompatible and connected by a covalent chemical bond. It is formulated as

$$E[u] = \int_{\mathbb{T}^d} \left[\frac{\epsilon}{2} |\nabla u|^2 + \frac{1}{\epsilon} W(u) \right] dx + \frac{\gamma}{2} \int_{\mathbb{T}^d} \left| (-\Delta)^{-\frac{1}{2}} \Big(f(u) - \omega \Big) \right|^2 dx + \frac{M}{2} \left[\int_{\mathbb{T}^d} (f(u) - \omega)\, dx \right]^2. \tag{1}$$

Here $\mathbb{T}^d = \prod_i [-X_i, X_i) \subset \mathbb{R}^d,\ d = 1, 2, 3$ is a periodic domain, and $u = u(x)$ is a phase field labeling function representing the density of A species with interfacial width ϵ. Function $W(u) = 18(u^2 - u)^2$ is a double well potential. The nonlinear function $f(u) = 6u^5 - 15u^4 + 10u^3$ keeps the 0-1 structure of u as $f(0) = 0$ and $f(1) = 1$. More importantly it also has the property that $f'(0) = f'(1) = 0$, which localize the force and avoid possible non-zeros and non-ones near the boundary of the interface [29] when considering L^2 gradient flow dynamics of (1). The first integral is a local surface energy, the second term represents the long-range interaction between the molecules with γ controlling its strength and $\omega \in (0, 1)$ being the fraction of species A, and the third term is a penalty term to fulfill the volume constraint

$$\int_{\mathbb{T}^d} (f(u) - \omega)\, dx = 0. \tag{2}$$

Received by the editors on July 14, 2022 and, accepted on May 30, 2023.
2000 *Mathematics Subject Classification.* XXX.
* Corresponding author.

Allen-Cahn-Ohta-Kawasaki(ACOK) model [29], which is the focus of this paper, takes the L^2 gradient dynamics of the OK free energy in (1) as

(3)
$$\begin{aligned} u_t =& -\frac{\delta E[u]}{\delta u} \\ =& \varepsilon\Delta u - \frac{1}{\varepsilon}W'(u) - \gamma(-\Delta)^{-1}(f(u)-\omega)f'(u) - M\left[\int_{\mathbb{T}^d}(f(u)-\omega)\,dx\right]f'(u). \end{aligned}$$

Here the long-range interaction term satisfies the zero-mean condition:

$$\int_{\mathbb{T}^d}(-\Delta)^{-1}(f(u)-\omega)dx = 0. \tag{4}$$

Various successful and efficient partial differential equation (PDE) solvers can be applied to the model, such as Finite Difference, Finite Element [20], Spectral method [10, 15], etc. In recent years, some novel numerical methods have also been studied for Cahn-Hilliard dynamics, i.e., the H^{-1} gradient flow dynamics of OK energy, such as the midpoint spectral method [1], and the Invariant Energy Quadratization (IEQ) method [5]. In [29], a first-order energy stable linear semi-implicit method is introduced and analyzed for the ACOK equation. Our goal in this paper is to study the approximation of the solution of the ACOK equation with neural networks.

Recently, there has been an increasing interest in solving PDEs with machine learning methods, among which the physics-informed neural networks(PINNs) [24] have gained tremendous popularity. PINNs are based on a fully-connected feed-forward deep neural network (DNN) [14, 26], which is often interpreted utilizing universal approximation theorem [6, 11, 12, 18]. The structure of a fully-connected DNN, consisting of an input layer, hidden layers, and an output layer, is shown in Figure 1. Every node in one layer is connected with every node in another by a series of computations. The nodes in the input layer are the specific data studied, and the nodes in the output are the expected outcome. For example, in the case of facial recognition, the input could be the RBG values (features) of each pixel of the sample pictures, while the output could be the assigned numerical values representing the individuals (classes). The nodes in the hidden layers are called neurons, and they are responsible for the performance of the neural network. There can be many hidden layers in a neural network; in the case of Figure 1, the number of hidden layers is three.

A column vector is fed to each node of the input layer. Then in the first hidden layer, we multiply it by another row vector, i.e., weight, to get the product. This is the case that there is only one neuron in the relevant hidden layer. If there are more neurons, the row vector will be a weight matrix instead, where the number of rows of the matrix represents the number of neurons. A bias vector is then added to the product, and an activation function is applied to the new vector, which contributes to the output of the current hidden layer, which is also the input of the next layer. In a nutshell, let M be the number of hidden layers, p be the input of the neural network, then the output a of the neural network will be a^M, with

$$a^0 = p, \tag{5}$$

$$a^i = f^i(W^i a^{i-1} + b^i),\ i = 1, ..., M-1, \tag{6}$$

where a^{i-1} is the input, W^i represents the weights, b^i represents the bias, and f^i is the activation function in the ith layer.

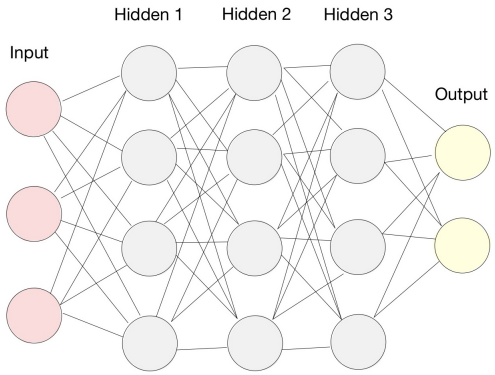


FIGURE 1. Fully-connected DNN structure.

To obtain the proper weights and bias, we perform an optimization process, for example, gradient descent, on the weight and bias to update them. To calculate the derivatives involved in the optimization process, the network goes through a process called back-propagation [25], which is essentially a chain rule applied on each layer starting from the final one. We start with a certain loss function:

$$\text{Loss}(y, a(\Theta)),$$

where y is the object, the network is approximating, and Θ denotes the weights and bias of the DNN. The loss function, depending on the specific problem, could be a mean square error (MSE), mean absolute error (MAE), log loss function, etc. We then perform back-propagation on the loss function, trace all the way back to get the derivatives of weights and bias for each layer, and then use the optimization tool to update the weights and bias. A gradient descent update is as follows:

$$\Theta^{n+1} = \Theta^n - \eta \nabla_\Theta \text{Loss},$$

where η is the learning rate, and ∇_ΘLoss is the direction of update. It works essentially like a directing system, telling you which direction to go and how far to go in that direction based on your current location and destination. The optimization process will be repeated several times until the prescribed accuracy is achieved. We call this number of repetitions the number of epochs.

Furthermore, there are a variety of optimizers that can be selected. The previously mentioned gradient descent is one of them. While it has many advantages, it does not work well in our non-convex problem due to its notoriously poor performance on escaping local minimums [7]. Therefore, other optimizers, such as momentum [23], ADAM [16], Adadelta [30], RMSprop [13], etc are developed to better address this issue. The momentum method adds an accelerator in the relevant direction. For example, think about rolling a ball down a ramp. As the ball goes down, it gains momentum and travels faster and faster. Other methods, like Adadelta, use different forms of adaptive learning rates. It is worth mentioning that RMSprop works by dividing "the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight" [13].

In our case, we adopt ADAM as our optimizer for each epoch and L-BFGS-B with a certain stopping condition as the optimizer afterward. Introduced in [16], ADAM can be viewed as a combination of the momentum method and RMSprop.

It uses the mean m_t and uncentered variance v_t of the gradient with regards to time t to adapt the learning rate for each weight and bias of the neural network:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$

where g_t is the gradient of the loss function at time step t, and the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ direct the decay rates of m_t and v_t. A bias correction is also applied to m_t and v_t, which leads to

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

and to update the weights and bias, we perform $\Theta_t = \Theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}$, where η is the learning rate, and ε is arbitrary. On the other hand, the L-BFGS-B optimizer [3] uses a "limited memory" BroydenCFletcherC GoldfarbCShanno (BFGS) [2,8,9,27] matrix to represent the Hessian matrix of the objective function, making it great for optimization problems with a large number of variables. Stems from the L-BFGS method [19,21], L-BFGS-B places upper and lower bound constraints on variables. It identifies fixed and free variables for each step with a basic gradient method. It then performs the L-BFGS method on the free variables to achieve better accuracy.

For PINNs, sampled collocation points are fed to the DNN, and the function value is returned to the output layer. Weight and bias in each layer can be learned or updated by optimizing the error function to reach the prescribed accuracy, in which the back-propagation technique is needed to compute the first-order derivatives. Different from traditional machine learning DNN, which requires a very large amount of ground truth data to make an accurate prediction, PINNs take advantage of the physics information and make up for the lack of sufficient ground truth data. It divides the error function into two components, one corresponding to the known ground truth data and the other corresponding to the physics information. Consequently, our goal is to minimize the error between the limited amount of ground truth and the prediction, as well as how much the prediction "fits" the physics information. Furthermore, as shown in [4], PINNs can be employed to solve inverse problems, in particular, inverse scatter problems in photonic metamaterials and nano-optics technologies, by transforming them into parameter retrieving problems. However, PINNs also have their downsides. As suggested in [17], "existing PINN methodologies can learn good models for relatively trivial problems", and it can fail when predicting solutions for a more complex model, which is likely due to the fact that the setup of PINNs makes it hard to optimize the less-smooth loss landscape. They have, therefore, proposed a "curriculum regularization", which is finding a good initialization for weights and training the model gradually. Notably, this has made the optimization process easier. They also mentioned a sequence-to-sequence learning method, which is similar to the Time-Adaptive Method proposed in [28]. This method can make learning easier by learning in small intervals.

Traditional machine learning approaches often rely on extensive labeled data to make accurate predictions, while mathematical methods may not fully capture the complexities of real-world data. By combining these two methodologies, Physics-Informed Neural Networks (PINNs) can achieve precise predictions even with limited data and effectively adapt the underlying equation to ground truth observations. Consequently, PINNs have the potential to address real-world challenges across diverse applications, such as physics, medicine, and fluid dynamics. It's important to note that using PINNs to discover solutions to the ACOK model is just the initial step. Our future objective is to leverage PINNs as a tool for learning the general kernel function governing long-range interactions, which traditional numerical

methods for PDEs struggle to solve. Although traditional numerical methods have proven successful and efficient, the unique advantages offered by PINNs should not be overlooked.

In this work, our main contribution lies in addressing the challenges posed by the complexity of the ACOK model when applied to PINNs. As shown in Equation 1, the ACOK model cannot be directly solved using the original PINNs due to several factors. These factors include the long-range interaction term $(-\Delta)^{-1}(f(u)-\omega)$, the volume constraint (2), and the zero-mean condition (4), all of which introduce nonlocal conditions that require computational and structural modifications to the basic PINNs framework. To handle the long-range interaction term, we incorporate a second neural network to solve for $(-\Delta)^{-1}(f(u)-\omega)$. In order to account for the volume constraint and the zero-mean condition, we introduce a third neural network, where the x variable is uniformly sampled and the t variable is randomly sampled.

The rest of this paper is organized as follows. In Section 2, we present the numerical methods. Specifically, we briefly revisit the baseline PINNs in Section 2.1. Then we elaborate on the modified PINNs, including the application of the periodic boundary condition, the approximation of the long-range interaction term, the volume constraint, and the zero-mean condition, in Section 2.2. We will then present and discuss some 1-dimensional results in Section 3. In the end, we give a detailed discussion and explain our future work in Section 4.

2. Numerical Methods

Our goal in this paper is to take the ground truth $u(0,x)$ data as input, and predict the solution $u(t,x)$ of ACOK equation with neural networks.

2.1. Baseline PINNs. For the basic setup, we follow the basic PINNs. Let us recall (1) and define the residual function $F(t,x)$ as

$$F(t,x) := u_t - \varepsilon\Delta u + \frac{1}{\varepsilon}W'(u) + \gamma(-\Delta)^{-1}(f(u)-\omega)\cdot f'(u) + M\left[\int (f(u)-\omega)\,dx\right]\cdot f'(u), \tag{7}$$

where $u(t,x)$ can be approximated by a deep neural network Net_u, and $F(t,x)$ can be calculated by a shared parameter neural network Net_F based on the predicted $u(t,x)$. The network can be learned by minimizing the mean square error:

$$\mathrm{MSE} = \mathrm{MSE}_u + \mathrm{MSE}_F,$$

where

$$\mathrm{MSE}_u = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|u(t_u^i,x_u^i)-u^i\right|^2, \quad \mathrm{MSE}_F = \frac{1}{N_F}\sum_{i=1}^{N_F}\left|F(t_F^i,x_F^i)\right|^2.$$

As shown in Figure 2, $\{t_u^i,x_u^i\}_{i=1}^{N_u}$ are selected from the collocation points that are along the boundary, i.e. the blue points, and at the initial time, i.e. the red points. Also, $\{t_F^i,x_F^i\}_{i=1}^{N_F}$ are sampled from the internal collocation points, i.e. the black points.

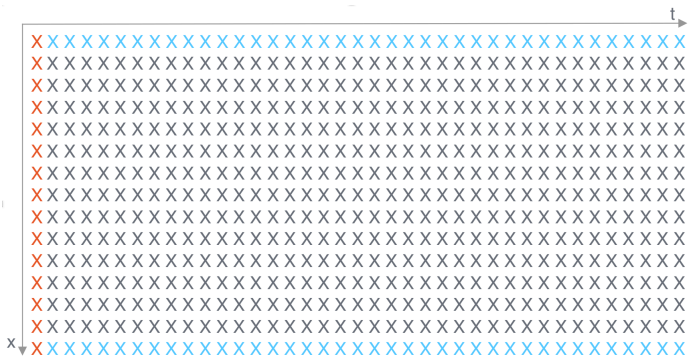


FIGURE 2. Sampling for the initial points, boundary points, and internal points.

For the purpose of computational efficiency and prediction accuracy, a random sample is taken from the initial and boundary collocation points. Moreover, the internal points are selected near-randomly using Latin Hypercube Sampling(LHS) method.

We take pairs of $\{t_u^i, x_u^i\}_{i=1}^{N_u}$ and $\{t_F^i, x_F^i\}_{i=1}^{N_F}$ as the input, and employ the feed forward deep neural network(DNN) to approximate the u value at these collocation points: $u(t_u^i, x_u^i)$ and $u(t_F^i, x_F^i)$. To check the accuracy of the prediction, we take the mean square error between the ground truth u^i and the prediction $u(t_u^i, x_u^i)$. Note that we take the mean square error of $F(t_F^i, x_F^i)$ obtained through Net_F to ensure that the approximation by the DNN satisfies the physics, which is described by the ACOK equation.

As previously mentioned, ADAM is selected as our optimizer for each epoch, and L-BFGS-B is employed in the last step. Both of them have built-in packages developed in Python for implementation. We also use tanh as the activation function for each hidden layer.

2.2. Modified PINNs. Next, we demonstrate how we develop the modified PINNs. There are several difficulties in applying the baseline PINNs directly to the ACOK equation, which motivate us to propose the modified PINNs to overcome these issues.

2.2.1. Periodic boundary conditions. As we have the periodic boundary condition in the ACOK equation, MSE_u is separated further into two components, one corresponding to the initial collocation points, i.e. MSE_{in}, and the other corresponding to the boundary collocation points, i.e. MSE_b. Hence, we can update the loss function as:

$$\text{MSE} = \text{MSE}_{in} + \text{MSE}_b + \text{MSE}_F,$$

where

$$\text{MSE}_{in} = \frac{1}{N_{in}} \sum_{i=1}^{N_{in}} \left| u(t_{in}^i, x_{in}^i) - u_0^i \right|^2, \quad \text{MSE}_b = \frac{1}{N_b} \sum_{i=1}^{N_{lb}} \left| u(t_{lb}^i, x_{lb}^i) - u(t_{ub}^i, x_{ub}^i) \right|^2,$$

and

$$\text{MSE}_F = \frac{1}{N_F} \sum_{i=1}^{N_F} \left| F(t_F^i, x_F^i) \right|^2.$$

Since we have the ground truth at the initial time, we take the mean square error between the u value predicted by the DNN, $u(t_{in}^i, x_{in}^i)$, and the ground truth u^i. However, the periodic boundary condition means that we will calculate MSE_b by taking the mean square error between the u value on the lower bound, i.e. $u(t_{lb}^i, x_{lb}^i)$, and the u value on the upper bound, i.e. $u(t_{ub}^i, x_{ub}^i)$, at the same time t, since the ground truth value of u is unknown.

2.2.2. Approximation of the inverse of Laplacian. One major difficulty is how to approximate the long-range interaction term in our model. As stated above, the purpose of the shared parameter neural network Net_F is to approximate $F(t_F^i, x_F^i)$ given $u(t_F^i, x_F^i)$. Recall (7):

$$F := u_t - \varepsilon\Delta u + \frac{1}{\varepsilon}W'(u) + \gamma(-\Delta)^{-1}(f(u)-\omega)\cdot f'(u) + M\left[\int (f(u)-\omega)\,dx\right]\cdot f'(u).$$

While u_t, Δu can be obtained by back-propagation, and $W'(u)$, $f(u)$ and $f'(u)$ can be directly calculated using the explicit expression, the difficulty lies with the approximation of $(-\Delta)^{-1}(f(u)-\omega)$ and $\int(f(u)-\omega)\,dx$.

To approximate the inverse of the Laplacian

$$(-\Delta)^{-1}(f(u)-\omega),$$

we add a second output ν to Net_u to approximate

$$\nu := (-\Delta)^{-1}(f(u)-\omega).$$

If $(-\Delta)$ operator is applied on both sides, we recover:

$$-\Delta\nu = f(u) - \omega.$$

Since ν has already been approximated by the DNN, $\Delta\nu$ can be calculated by back-propagation. To ensure the accuracy of ν, a mean square error can be taken between $(-\Delta)\nu$ and $f(u)-\omega$, as the latter is achieved by the explicit expression:

$$\mathrm{MSE}_{Lap} = \frac{1}{N_F}\sum_{i=1}^{N_F}\left|-\Delta\nu(t_F^i, x_F^i) - (f(u(t_F^i, x_F^i)) - \omega)\right|^2,$$

and the first four terms of F can be calculated by:

$$F = u_t - \varepsilon u_{xx} + \frac{1}{\varepsilon}W'(u) + \gamma\nu\cdot f'(u).$$

Furthermore, as the ground truth of $(-\Delta)^{-1}u_{in}$ and the periodic boundary condition on $(-\Delta)^{-1}u$ are available, the total mean square loss is modified as:

$$\mathrm{MSE} = (\mathrm{MSE}_{u_{in}} + \mathrm{MSE}_{((-\Delta)^{-1}u)_{in}}) + (\mathrm{MSE}_{u_b} + \mathrm{MSE}_{((-\Delta)^{-1}u)_b}) + \mathrm{MSE}_F + \mathrm{MSE}_{Lap},$$

where

$$\mathrm{MSE}_{((-\Delta)^{-1}u)_{in}} = \frac{1}{N_{in}}\sum_{i=1}^{N_{in}}\left|(-\Delta)^{-1}u(t_{in}^i, x_{in}^i) - ((-\Delta)^{-1}u)_0^i\right|^2,$$

$$\mathrm{MSE}_{((-\Delta)^{-1}u)_b} = \frac{1}{N_b}\sum_{i=1}^{N_{lb}}\left|(-\Delta)^{-1}u(t_{lb}^i, x_{lb}^i) - (-\Delta)^{-1}u(t_{ub}^i, x_{ub}^i)\right|^2.$$

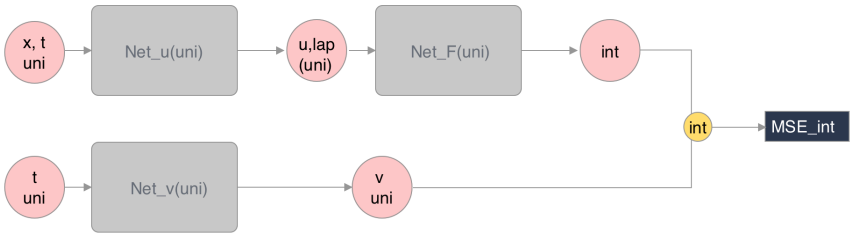
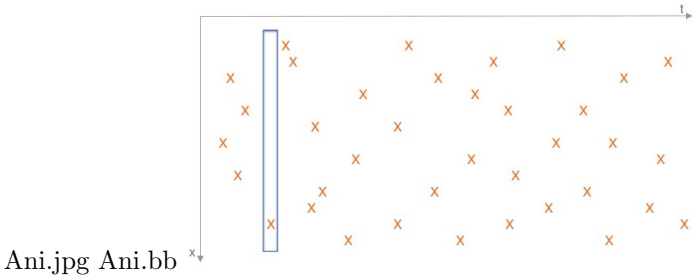


FIGURE 3. Net_v Structure.

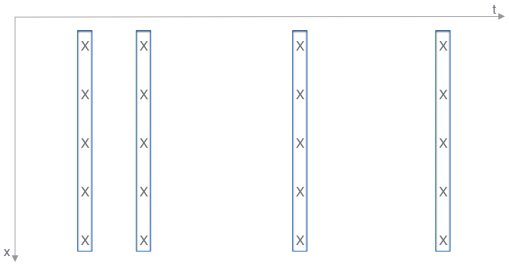
2.2.3. Integral approximation. Next, we propose a structural modification to approximate the integral term. The integral term

$$\int (f(u) - \omega)\, dx,$$

imposes a problem to the modified PINNs, as the near-randomly sampled collocation points are not evenly distributed along each t. It is usually the case that at some t there are one or no points selected, shown in Figure 4. As a result, the integral can not be calculated accurately for each t. To resolve this issue, a separate shallow neural network Net_v is introduced, which only takes t as input and has a built-in uniform mesh on x. This network aims to output the value of the integral term for each randomly sampled t. After the network is trained, the updated optimal weights and bias ensure that for any input t, the network will produce the corresponding $\int (f(u) - \omega)\, dx$, even if the new input t is different from the t used for training.



(a) LHS Sampling.



(b) Uniform x and Random t Mesh.

FIGURE 4. Collocating point sampling strategy: (a) LHS sampling; (b) uniform spatial sampling and random t.

As demonstrated in Figure 3, to train Net_v, the mean square error between the output of Net_v and the discrete form of the integral

$$\sum_{i=1}^{N_x} f(u(t^j_{uni}, x^i_{uni})) \cdot \Delta x - \omega,$$

is taken:

$$\text{MSE}_{int} = \frac{1}{N_t} \sum_{j=1}^{N_t} \left| v(t^j_{uni}) - \left[\sum_{i=1}^{N_x} f(u(t^j_{uni}, x^i_{uni})) \cdot \Delta x - \omega \right] \right|^2,$$

where $\{t^j_v, x^i_v\}$ denotes the new uniform x and random t mesh as in Figure 4, and $u(t^j_{uni}, x^i_{uni})$ can be output by Net_u with the same set of input. Furthermore, $\sum_{i=1}^{N_x} f(u(t^j_{uni}, x^i_{uni}))$ is obtained by taking the column sum of output of $f(u(t^j_{uni}, x^i_{uni}))$ along each sampled t direction.

After the optimal weight and bias are achieved, the t part of the original LHS sampled points is fed to Net_v, and outputs

$$v = \int (f(u(t^i_F, x^i_F)) - \omega)\, dx,$$

and F can be approximated by

$$F = u_t - \varepsilon u_{xx} + \frac{1}{\varepsilon} W'(u) + \gamma\nu \cdot f'(u) + Mv \cdot f'(u),$$

which is calculated in Net_F. Taking the mean square error F will obtain the complete MSE_F.

Therefore, the total mean square loss is modified as:

$$\begin{aligned} \text{MSE} = {} & (\text{MSE}_{u_{in}} + \text{MSE}_{((-\Delta)^{-1}u)_{in}}) + (\text{MSE}_{u_b} + \text{MSE}_{((-\Delta)^{-1}u)_b}) \\ & + \text{MSE}_F + \text{MSE}_{Lap} + \text{MSE}_{int}, \end{aligned}$$

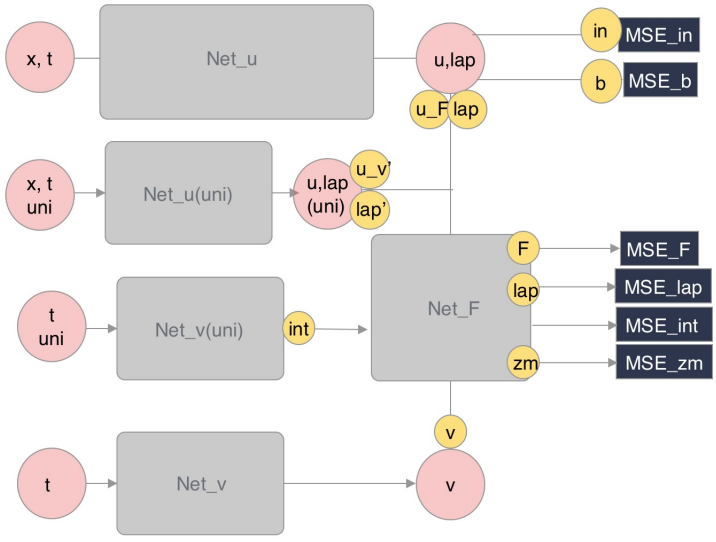


FIGURE 5. The modified PINNs structure with Net_v and zero-mean condition.

where

$$\mathrm{MSE}_F = \frac{1}{N_F}\sum_{i=1}^{N_F}\left|F(t_F^i, x_F^i)\right|^2.$$

2.2.4. Zero-mean constraint. For the zero-mean condition on the inverse of the Laplacian, since the calculation of the mean is involved, the uniform x and random t meshgrid created for the integral approximation can be used as input of Net_u, and the column sum of the output $(-\Delta)^{-1}u(t_F^i, x_F^i)$, which corresponds to each time t, is minimized in a loss function

$$\mathrm{MSE}_{zm} = \frac{1}{N_t}\sum_{j=1}^{N_t}\left|\sum_{i=1}^{N_x}(-\Delta)^{-1}u(t_F^i, x_F^i)\cdot\Delta x\right|^2.$$

The complete modified PINNs structure is shown in Figure 5.

Hence, the modified PINNs is learned by minimizing the mean square loss:

$$\begin{aligned}\mathrm{MSE} = {} & (\mathrm{MSE}_{u_{in}} + \mathrm{MSE}_{((-\Delta)^{-1}u)_{in}}) + (\mathrm{MSE}_{u_b} + \mathrm{MSE}_{((-\Delta)^{-1}u)_b}) + \mathrm{MSE}_F \\ & + \mathrm{MSE}_{Lap} + \mathrm{MSE}_v + \mathrm{MSE}_{zm}.\end{aligned}$$

Moreover, since certain components have more importance over the others, such as the mean square error on the initial condition for its ground truth availability, adjustable weights are applied to each component:

$$\begin{aligned}\mathrm{MSE} = {} & W_{u_{in}}\mathrm{MSE}_{u_{in}} + W_{((-\Delta)^{-1}u)_{in}}\mathrm{MSE}_{((-\Delta)^{-1}u)_{in}} \\ & + W_{u_b}\mathrm{MSE}_{u_b} + W_{((-\Delta)^{-1}u)_b}\mathrm{MSE}_{((-\Delta)^{-1}u)_b} \\ & + W_F\mathrm{MSE}_F + W_{Lap}\mathrm{MSE}_{Lap} + W_{int}\mathrm{MSE}_{int} + W_{zm}\mathrm{MSE}_{zm}.\end{aligned}$$

3. Numerical Results

Now we have all the pieces regarding how to apply PINNs on ACOK. We can combine them and implement them in Python. The code is based on the work of Maziar Raissi et al. [24]. We now show the results of the modified PINNs on 1-dimensional ACOK with different time intervals, ranging from 1×10^{-3}s to 1×10^{-2}s. Fine-tuning on hyper-parameter is needed to reach the best results, but it is out of our current research scope.

Figure 6 are run for 495 epochs, with weights $W_{u_{in}} = 1\times10^5$, $W_{((-\Delta)^{-1}u)_{in}} = 5\times10^6$, $W_{u_b} = 1$, $W_{((-\Delta)^{-1}u)_b} = 30$, $W_F = 1$, $W_{Lap} = 500$, $W_{int} = 1$, $W_{zm} = 30$. There are 10 hidden layers in Net_u, each containing 20 neurons, while only 3 hidden layers with 10 neurons for each layer were needed for Net_v, since it is a much simpler neural network.

We can see that the modified PINNs perform remarkably well in smaller time periods. However, as the time interval increases, the prediction of $(-\Delta)^{-1}u$ at the right tail is getting notably worse. There are several ways to address this problem. One approach is the Time-Adaptive Method [28], or similarly, the sequence-to-sequence learning [17], which learns only one small-time period at a time and then uses the prediction as to the initial data for the next period. Since our model approximates well with short time intervals, we can compile all the sub-results together to get the entire solution. For example, we could use the data of the last time column in Figure 6(a) as the initial data for the next 100 columns and repeat the process in 100 column increments to achieve good results.

Furthermore, the problem can be improved by rescaling the sampling points. As shown in Figure 7(a), changes in phase separation occur most rapidly during the earlier time period. Therefore, we propose rescaling the sampling points to

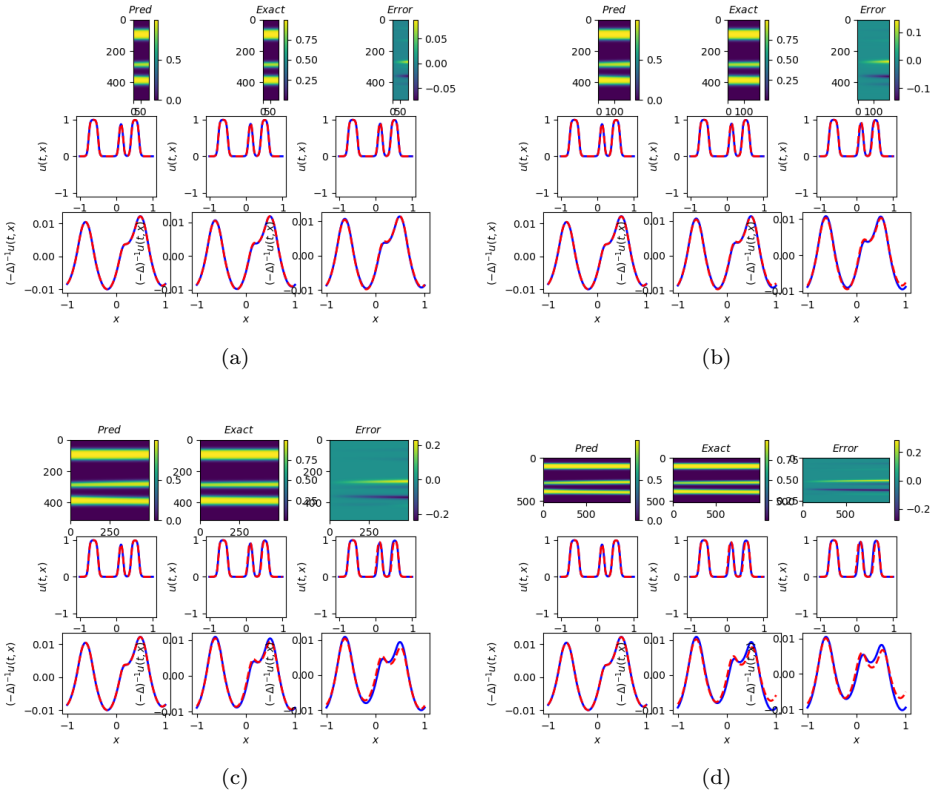


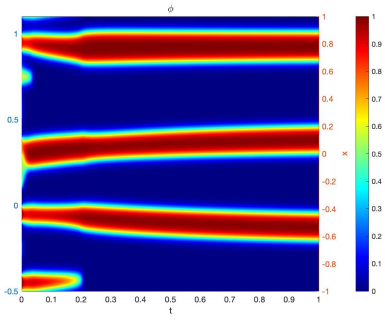
FIGURE 6. Numerical results of the ACOK equation using the modified PINNs with various settings. (a) $t = 1 \times 10^{-3}$, $N_0 = 500$, $N_b = 95$, $N_f = 2 \times 10^4$, $N_{t_{uni}} = 20$; (b) $t = 2 \times 10^{-3}$, $N_0 = 500$, $N_b = 195$, $N_f = 4 \times 10^4$, $N_{t_{uni}} = 40$; (c) $t = 5 \times 10^{-3}$, $N_0 = 500$, $N_b = 495$, $N_f = 1 \times 10^5$, $N_{t_{uni}} = 100$; (d) $t = 1 \times 10^{-2}$, $N_0 = 500$, $N_b = 995$, $N_f = 2 \times 10^5$, $N_{t_{uni}} = 200$.

be denser in earlier time columns. Recall Figure 4, we continue to use the LHS sampling, but rescale the points polynomially to achieve the denser in the beginning effect, as in Figure 7(b).

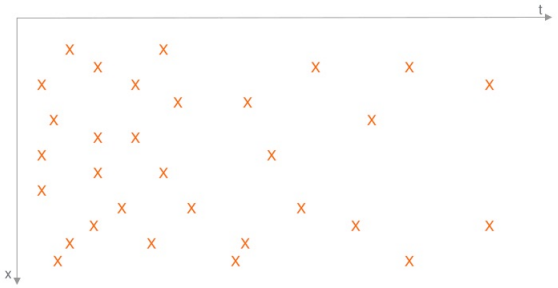
The results are shown as follows, Figure 8(a) is the result with original LHS sampling, with time interval being 3×10^{-3}s. Notice that the right tail on $(-\Delta)^{-1}u$ fits quite badly in this case due to the long time interval. Figure 8(b) is the result with a x^2 rescaling on the points. Although it does not fix the problem entirely, we can see that the tail on $(-\Delta)^{-1}u$ fits much better with a x^2 rescaling. In some cases, x^2 rescaling does not improve the result much. Therefore, x^3 rescaling could potentially be used to improve the fitting further.

Next, we present the results of the modified PINNs on 2D ACOK with different time intervals, ranging from 4×10^{-3}s to 2×10^{-2}s.

Figure 9 are run for 10000 epochs, with weights $W_{u_{in}} = 1 \times 10^5$, $W_{((-\Delta)^{-1}u)_{in}} = 5 \times 10^6$, $W_{u_b} = 1$, $W_{((-\Delta)^{-1}u)_b} = 30$, $W_F = 1$, $W_{Lap} = 500$, $W_{int} = 1$, $W_{zm} = 30$. There are 9 hidden layers in Net_u, each containing 32 neurons, while only 3 hidden layers with 10 neurons for each layer were needed for Net_v, same as in the 1D case.

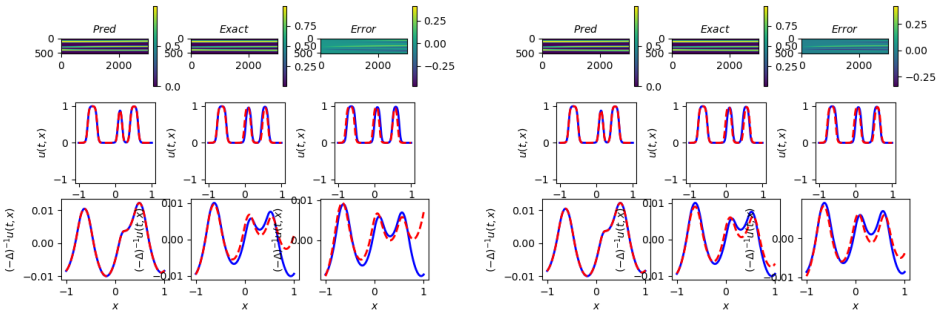


(a)



(b)

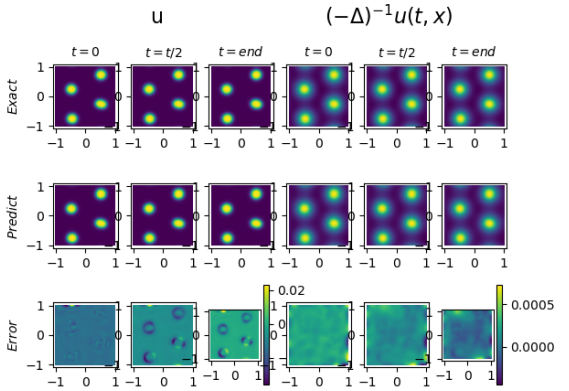
FIGURE 7. An illustration on how to improve the modified PINNs. (a) numerical results of the 1D ACOK equation with spectrum method; (b) scaling sample points.



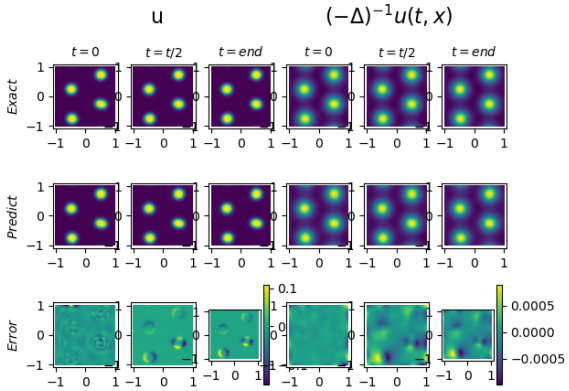
(a) Original LHS sampling.

(b) x^2 rescaling sampling.

FIGURE 8. A comparison of different sampling strategy. (a) shows the original LHS sampling; (b) shows the x^2 rescaling sampling. In both cases, we use $t = 3 \times 10^{-2}$, $N_0 = 500$, $N_b = 2995$, $N_f = 6 \times 10^5$, and $N_{t_{uni}} = 10$.



(a)



(b)

FIGURE 9. 2D Numerical approximations of the ACOK equation using the modified PINNs with various setups. (a) $t = 4 \times 10^{-3}$, $N_0 = 1664$, $N_b = 258$, $N_f = 33282$, $N_{t_{uni}} = 5$; (b) $t = 2 \times 10^{-2}$, $N_0 = 1664$, $N_b = 1290$, $N_f = 166410$, $N_{t_{uni}} = 25$.

Note that, similar to the 1D case, our modified PINNs in 2D perform well only within a small time period. As the time interval increases, the modified PINNs start to exhibit large errors, as shown in the third columns of (a) and (b) in Figure 9. Additionally, it is important to mention that in the 2D simulation, we simply use a near-equilibrium configuration as the initial condition, resulting in nearly identical solution snapshots at different times. However, our framework still encounters challenges in reproducing solutions when a more random initial condition is chosen. We acknowledge this limitation and will consider it in the future work.

4. Discussion and future work

During implementation, we have discovered that our modified PINNs are challenging to train due to their many hyper-parameters. Some hyper-parameters, such as learning rate, number of hidden layers, and number of neurons of each layer, are

fixed during our training. However, the weights, number of epochs, and number of sampling points need fine-tuning to achieve the best results.

The most challenging part of our fine-tuning is finding a well-balanced set of weights. Each weight controls a different part of the estimation. For instance, $W_{u_{in}}$, $W_{((-\Delta)^{-1}u)_{in}}$ controls the fitting of initial data, which is undoubtedly an essential part of our loss function since the initial data is the only ground truth data we have. But they differ from each other, as we later find out that the scale of $((-\Delta)^{-1}u)_{in}$ is 100 times smaller than that of u_{in}. Therefore, we also consider that by increasing $W_{((-\Delta)^{-1}u)_{in}}$ to counter the scale issue. A similar scaling technique is applied to W_{u_b} and $W_{((-\Delta)^{-1}u)_b}$, which controls the periodic boundary conditions.

W_F dominates the physics information in our model, while W_{Lap} controls the accuracy of the approximation of the long-range interaction term. We have learned that the accuracy of the long-range interaction term is one of the key players in our approximation. W_{int} directs the volume constraint and, therefore, can be increased if the predicted result has an incorrect volume. W_{zm} applies only to the $((-\Delta)^{-1}u)_{in}$, which means the scaling problem also needs to be taken into consideration. We have also discovered that the interval of a number of epochs is quite important. The model will not train well with too few epochs. However, if we boost the number of epochs to the scale of the thousands, the model is over-trained and does not perform well.

Moreover, we tune the number of the sampling points using the principle that we want to keep the randomness while at the same time preserving as much of the known information as possible. We also remain attentive to the efficiency of the program. Again, the initial data, being the only piece of ground truth information available, need to have the most sampling points proportionally. We do not take all the initial data because we want to keep the randomness. We also sample a relatively large percentage of the boundary points for the same reason. As the total number of the grid points is quite large for the internal points, we cannot take as many as the sampling points while keeping the program's efficiency. Randomness also plays a crucial part here. Therefore, we only select less than half of the points. The number of randomly selected t columns in the uniform mesh, $N_{t_{uni}}$, follows the same principle.

To increase the randomness, we also implemented a mini-batch method. The idea is that instead of considering all the points simultaneously, we consider a small subset of the points at a time. In theory, it should need fewer epochs and converge faster. However, as the computation in our case is quite complex, we have discovered that the mini-batch method takes significantly longer time than the regular approach.

In the 2D case of this problem, we encounter significant difficulty due to the exponential growth in the number of points and the introduction of the front and back bound in addition to the existing upper and lower bound. This increased complexity leads to insufficient memory capacity, further exacerbated by a more intricate back-propagation process that incorporates the second-order derivative. To tackle these challenges, we have implemented a two-pronged approach. Firstly, we fine-tune the sampling ratio to achieve an optimal balance between computational resources and accuracy. This allows us to effectively manage memory usage without compromising the integrity of our calculations. Secondly, we maintain a relatively small time interval, which reduces the overall complexity of the problem and ensures a more manageable and efficient computational process.

There are several directions in which the current work can be extended. First, as we mentioned in the Introduction, this work is our first attempt to use machine learning tools to discover the solutions to ACOK model. An ongoing work is to use PINNs to learn the kernel structures for a general long-range interaction term, given multiple solution snapshots. This ongoing work, using PINNs to predict the underlying physics, is indeed the more important aspect of the application of PINNs to ACOK model, comparing to discovering the solutions. Second, it is well known that ACOK dynamics (more generally, the phase field equations) highly depends on the initial states. As the randomness of the initial states increases, the ACOK dynamics to system equalibira become more complex. We will explore systematically the PINNs model for more random initials, particularly in the 2D case, in order to capture the complete ACOK dynamics (or at least the initial stage which involves the most significant change on the solution configuration). Also, the efficiency of the training process can be further improved. Although the current 1D and 2D results are obtained within an hour, initializing the randomized weights and biases closer to the optimal solution could significantly reduce the number of required epochs, and consequently, the overall training time. Furthermore, it would be valuable to explore alternative machine learning approaches, such as Fourier Neural Operators (FNO), conventional neural networks, and transformer-based methods, for their potential applicability to ACOK model.

Acknowledgements

The work of J. Xu and Y. Zhao is supported by a grant from the Simons Foundation through Grant No. 357963 and NSF grant DMS-2142500. J. Zhao would like to acknowledge the support from National Science Foundation, United States, with grant DMS-2111479.

References

[1] Barbora Beneov, Christof Melcher, and Endre Sli. An implicit midpoint spectral approximation of nonlocal Cahn–Hilliard equations. *SIAM Journal on Numerical Analysis*, 52:1466–1496, 01 2014.

[2] C. G. BROYDEN. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970.

[3] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[4] Yuyao Chen, Lu Lu, George Karniadakis, and Luca Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express*, 28, 03 2020.

[5] Qing Cheng, Xiaofeng Yang, and Jie Shen. Efficient and accurate numerical schemes for a hydro-dynamically coupled phase field diblock copolymer model. *J. Comput. Phys.*, 341:44–60, 2017.

[6] George Cybenko. Approximation by superpositions of a sigmoidal function. Technical report, Department of computer Science, Tufts University, Medford, MA, October 1988.

[7] Yann N. Dauphin, Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014.

[8] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 01 1970.

[9] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

[10] David Gottlieb and Steven A. Orszag. *Numerical Analysis of Spectral Methods*. Society for Industrial and Applied Mathematics, 1977.

[11] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, 1995.

[12] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[13] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning, lecture 6a overview of mini-batch gradient descent, 2012.
[14] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
[15] M. Yousuff. Hussaini, Thomas A. Zang, Institute for Computer Applications in Science, and Engineering. *Spectral methods in fluid dynamics*. Institute for Computer Applications in Science and Engineering, NASA Langley Research Center ; National Technical Information Service, 1986.
[16] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.
[17] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *CoRR*, abs/2109.01050, 2021.
[18] Kurt and Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.
[19] Hermann G. Matthies and Gilbert Strang. The solution of nonlinear finite element equations. *International Journal for Numerical Methods in Engineering*, 14:1613–1626, 1979.
[20] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, 2 edition, 2005.
[21] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.
[22] Takao Ohta and Kyozi Kawasaki. Equilibrium morphology of block copolymer melts. *Macromolecules*, 19:2621–2632, 1986.
[23] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, January 1999.
[24] Maziar Raissi, Paris Perdikaris, and George Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 11 2018.
[25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.
[26] David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, Cambridge, MA, USA, 1986.
[27] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.
[28] Colby L. Wight and Jia Zhao. Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. *CoRR*, abs/2007.04542, 2020.
[29] Xiang Xu and Yanxiang Zhao. Energy stable semi-implicit schemes for AllenCahnOhtaKawasaki model in binary system. *Journal of Scientific Computing*, 80, 09 2019.
[30] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Department of Mathematics, George Washington University, Washington, DC, USA
E-mail: jix29@gwmail.gwu.edu (J. Xu)

Department of Mathematics & Statistics, Utah State University, Logan, UT, USA.
E-mail: t jia.zhao@usu.edu (J. Zhao)

Department of Mathematics, George Washington University, Washington, DC, USA
E-mail: yxzhao@email.gwu.edu (Y. Zhao)