

Solving Allen-Cahn and Cahn-Hilliard Equations using the Adaptive Physics Informed Neural Networks

Colby L. Wight¹ and Jia Zhao^{1,*}

¹ Department of Mathematics & Statistics, Utah State University, Logan, UT, USA.

Received 12 May 2020; Accepted (in revised version) 21 July 2020

Abstract. Phase field models, in particular, the Allen-Cahn type and Cahn-Hilliard type equations, have been widely used to investigate interfacial dynamic problems. Designing accurate, efficient, and stable numerical algorithms for solving the phase field models has been an active field for decades. In this paper, we focus on using the deep neural network to design an automatic numerical solver for the Allen-Cahn and Cahn-Hilliard equations by proposing an improved physics informed neural network (PINN). Though the PINN has been embraced to investigate many differential equation problems, we find a direct application of the PINN in solving phase-field equations won't provide accurate solutions in many cases. Thus, we propose various techniques that add to the approximation power of the PINN. As a major contribution of this paper, we propose to embrace the adaptive idea in both space and time and introduce various sampling strategies, such that we are able to improve the efficiency and accuracy of the PINN on solving phase field equations. In addition, the improved PINN has no restriction on the explicit form of the PDEs, making it applicable to a wider class of PDE problems, and shedding light on numerical approximations of other PDEs in general.

AMS subject classifications: (or PACs) To be provided by authors

Key words: Phase field, Allen-Cahn equation, Cahn-Hilliard equation, deep neural networks, physics informed neural networks.

1 Introduction

Phase field models have been widely embraced in the past few decades to study various problems in science and engineering, taking the applications in image analysis, material science, engineering, fluid mechanics, and life science as examples. Among them, two

*Corresponding author. Email addresses: colbywight5@gmail.com (C. L. Wight), jia.zhao@usu.edu (J. Zhao)

fundamental equations are the Allen-Cahn (AC) equation and Cahn-Hilliard (CH) equation, which are originally introduced to describe the non-conservative and conservative phase variables in the phase separation process, respectively. Both models are recognized as gradient flow systems, for which there exists a Lyapunov function, known as the free energy. From a modeling view, given a specified Lyapunov function, or a free energy function, the Allen-Cahn type equations can be derived as the L^2 gradient flow, and the Cahn-Hilliard type equations can be derived as the H^{-1} gradient flow, respectively. This generality makes the AC and CH type equations extremely useful in modeling many interfacial or multiphase problems. And many well-known PDE models turn out to be their special cases.

Given the nonlinearity in phase field equations, along with the stiff terms due to the small parameters, how to design accurate, efficient, and stable numerical algorithms for their numerical approximations have been intensively studied in the literature. Here is some literature that attracts our attention [7, 8, 11, 12, 18, 28, 29, 32, 33]. Interested readers are encouraged to read them and the references therein for further information. In this paper, we focus on a new numerical approximation approach by using the deep neural network. Our major goal is to investigate strategies to improve the capabilities of deep neural networks on solving phase field models, in particular, the Allen-Cahn equation and the Cahn-Hilliard equation.

The artificial neural network is named after the fundamental unit of computation inside the mammalian brain [20]. Many neurons inside the brain work together to carry out complex tasks. Similarly, an artificial neural network is composed of multiple connected neurons that work to solve complex tasks. A single neuron in a neural network can take input from multiple neurons (or nodes). Each input has a parameter called a weight associated with it. There is also typically a bias term that doesn't have an input associated with it. The neuron receives the sum of these inputs multiplied by their weights, along with added bias. This weighted sum then goes through an activation function that gives the final output for this neuron. In the brain, a neuron usually doesn't fire unless the total of its input reaches a certain threshold. The output is either on or off. In deep learning continuous activation functions are more commonly used [21]. The *sigmoid* function can be used as a smoother version of the step function. There are benefits in using differentiable functions like this that will help in "learning" good weights. Other useful activation functions used in deep learning include *relu*, *tanh*, leaky *relu* [31] and *swish* [25]. A typical feed-forward, fully connected neural network has input going to and from multiple neurons. The input to the network makes up the input layer. The neurons of the input layer are then sent to other layers of neuron connections called hidden layers, and finally to the last layer, the output layer. See Fig. 1 for a representation of a simple neural network architecture.

Mathematically, the feed-forward neural network could be defined as compositions of nonlinear functions. Given an input $x \in \mathbb{R}^{n_1}$, and denote the output of the l -th layer as $a^{[l]} \in \mathbb{R}^{n_l}$, which is the input for $l+1$ -th layer. In general, we can define the neural network

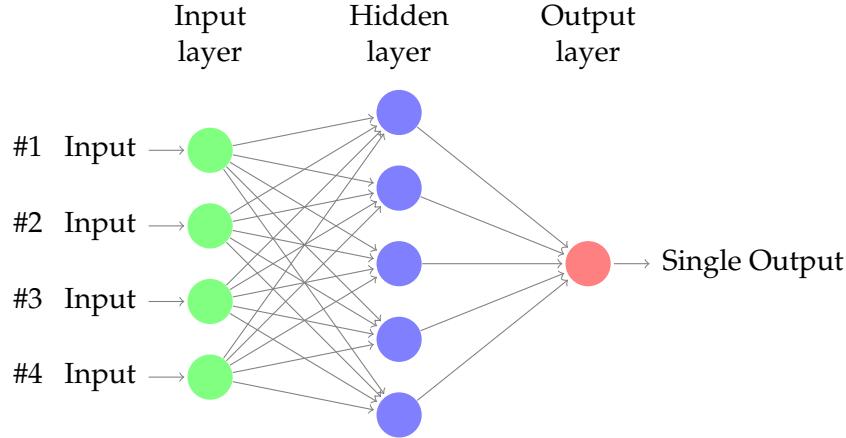


Figure 1: A diagram of a neural network with an input layer, hidden layer, and output layer. Each input gets sent to each neuron in the hidden layer. The arrows between the neurons all have a weight associated with them. The bias for each hidden neuron and output neuron are not shown.

as [15]

$$\begin{aligned} a^{[1]} &= x \in \mathbb{R}^{n_1}, \\ a^{[k]} &= \sigma(W^{[k]} a^{[k-1]} + b^{[k]}) \in \mathbb{R}^{n_k}, \quad \text{for } k=2,3,\dots,L, \end{aligned} \quad (1.1)$$

where $W^{[k]} \in \mathbb{R}^{n_k \times n_{k-1}}$ and $b^{[k]} \in \mathbb{R}^{n_k}$ denote the weights and biases at layer k respectively, σ denotes the activation function. Essentially neural networks are non-linear mappings with many parameters. Due to a large number of parameters, they are referred to as a "black-box" [6]. A loss function is defined to measure the difference between the output of the network with the known (expected) output. Then, the parameters (the weights and biases) can be "learned" by minimizing the loss function. Typically a gradient-based optimization scheme is applied. The backpropagation algorithm is an efficient way to find the gradient of this highly dimensional loss function [14]. Stochastic gradient descent is a popular method that uses smaller subsets of the training data called batches for each step to achieve better results [4].

In the past few years, there has been intensive research on understanding how deep neural networks can be adopted to solve and discover differential equations [1–3, 9, 10, 17, 19, 22–24, 26, 30]. For instance, various studies deal with problems of data-driven modeling, especially ones using deep learning to solve differential equations [13]. Others propose approaches to use existing experiment data to identify the differential equations themselves in order to model real-world phenomenon [5, 27]. In particular, Raissi et al. propose the physics informed neural network (PINN) to aid in both the solution of differential equations as well as their discovery [24]. The PINN was shown to solve Burgers' equation and the Schrodinger equation with certain initial conditions accurately. Though the PINN has been widely appreciated in the community, we found it is not capable of solving the phase field equations in many cases properly. As a result, we observe that

some parts of solutions (both spatially and temporally) for the phase field models might be harder to learn than other parts. These difficult areas could even change over the course of learning the solution. This is reasonable given the solutions for phase field models usually have sharp transition layers, and the layers evolve with time. Unfortunately, the original PINN method [24] failed to address these issues, as it involves choosing training data points at the beginning of training randomly across the domain and fixes them over the entire training process.

This motivates us to conduct the research in this paper. This major goal of this paper is to investigate strategies to improve the approximation capabilities of physics informed deep neural networks on solving the phase field equations. We propose strategies to improve the accuracy and efficiency of PINNs. The major contribution of this paper is to involve the idea of adaptive sampling data points over the course of training, both with time and space position in mind. In the rest of this paper, we will introduce our ideas in detail, and provide several numerical examples to justify our approach further.

2 Improved physics informed neural networks

In this section, we will first give a brief review of the physics informed neural network (PINN) [24]. Then, we propose several strategies to improve the accuracy and approximating capability of the PINN.

2.1 Physics informed neural networks

To elucidate the idea of the physics informed neural network (PINN) [24], we use the following Burgers' equations as an example, shown as

$$\begin{aligned} u_t + uu_x - (0.01/\pi)u_{xx} &= 0, \quad x \in [-1,1], \quad t \in [0,1], \\ u(0,x) &= -\sin(\pi x), \\ u(t,-1) &= u(t,1) = 0. \end{aligned} \tag{2.1}$$

To solve the Burgers' equation, the authors [24] introduce two neural networks: the u -network as

$$\mathcal{U}: (x,t) \rightarrow \mathcal{U}(x,t), \tag{2.2}$$

and the f -network as

$$\mathcal{F}: (x,t) \rightarrow \mathcal{U}_t(x,t) + \mathcal{U}(x,t)\mathcal{U}_x(x,t) - (0.01/\pi)\mathcal{U}_{xx}(x,t). \tag{2.3}$$

In this case, both the u and f networks have two input neurons, and one output in the final (output) layer. Given the initial data points $\{(0, x_u^i, u^i)\}_{i=1}^{N_u}$ and random samples from the boundary $\{(t_b^i, -1), (t_b^i, 1)\}_{i=1}^{N_b}$ and collocation points $\{(t_f^i, x_f^i)\}_{i=1}^{N_f}$ in the interior

of the domain $[-1,1] \times [0,1]$, the loss function for this problem is defined as the mean squared error of the u -network plus the mean squared error of the f -network, i.e.

$$MSE = MSE_u + MSE_b + MSE_f, \quad (2.4)$$

where the three terms are defined as

$$\begin{aligned} MSE_u &= \frac{1}{N_u} \sum_{i=1}^{N_u} |\mathcal{U}(0, x_u^i) - u^i|^2, \\ MSE_b &= \frac{1}{N_b} \sum_{i=1}^{N_b} |\mathcal{U}(t_u^i, 1)|^2 + |\mathcal{U}(t_u^i, -1)|^2, \\ MSE_f &= \frac{1}{N_f} \sum_{i=1}^{N_f} |\mathcal{F}(t_f^i, x_f^i)|^2, \end{aligned} \quad (2.5)$$

where N_u , N_b and N_f are the number of initial training data, boundary training data, and interior collocation points respectively. Here $(0, x_u^i)$ are the initial condition points that serve as the inputs to the u -network and u^i is the actual solution value of u at those points (provided in the initial condition). t_f^i and x_f^i are the collocation points passed into the f -network. Given the loss function value in (2.4) is small enough, the u -network approximates the solution well.

2.2 Some strategies to improve the accuracy of the physics informed neural networks

Though the PINN is shown to be powerful to solve the Burgers' equation, we find that a direct application of the PINN on solving the phase field equations would not provide accurate solutions in many cases. This motivates us to seek some extra techniques to improve the accuracy of the PINN. In this section, we introduce some strategies to improve the approximation power of the PINN.

2.2.1 Adding weights in the loss function

One simple technique to improve the learning capability of the PINN for solving phase field models is to add weights in the loss function. This is motivated by the fact that the phase field equations are dissipative, i.e., not reversible. For example, the Allen-Cahn equation, as with other reactive diffusion equations, can only be solved in the forward time direction. In other words, if the PINN fails to learn the solution at time $t = t_1$ well, there is little hope of learning the solution at a later time $t = t_2$ (with $t_2 > t_1$) accurately. In order to put an emphasis on the importance of first learning the solution near $t = 0$, we put more weight on MSE_u to enforce the u -neural network satisfies the initial condition. Specifically, we can redefine the loss function in (2.4) as follows:

$$MSE = C_0 MSE_u + MSE_b + MSE_f, \quad (2.6)$$

where C_0 is a big positive constant (that is tunable). Here we chose $C_0 = 100$ in this paper, if not specified separately.

2.2.2 Mini-batching strategy to improve convergence

Mini-batching is a technique that has been used in deep learning to improve performance. Instead of using the entire data set to calculate the exact direction of the gradient, a subset of the data called a batch or mini-batch is used to evaluate the direction. Mini-batching has been shown to help avoid less desirable local minimum better than full-batch gradient descent [16].

We point out that, in [24] and many follow-up papers, the authors do not use a mini-batching strategy in their training process. In this paper, we investigate the mini-batching approach and observe the mini-batch approach indeed can facilitate the convergence of the trained neural network for approximating phase field equations.

2.3 Adaptive strategies to improve the accuracy of the physics informed neural networks

In this section, we borrow the idea of temporal and spatial adaptivity in classical numerical methods for solving differential equations, and introduce some adaptive strategies to improve the accuracy of the physics informed neural networks.

2.3.1 Adaptive sampling of collocation points

Instead of using fixed sample points over the training process, we realize an adaptive resampling of the collocation points across the domain during the training process is essential for certain situations. In particular, for the phase field equations, there are moving interfaces that are sharp transitions over space, where finer meshes are desired to capture the evolution dynamics. Therefore, instead of only sampling points evenly across the domain, we periodically stop training and re-evaluate where points are needed most. We notice there is a correlation between the points that had a larger error in the u -network (the solution) and the points that had large errors in the f -network. This motivates us to use the error of f -network as an indicator for resampling.

In practice, we first train the network using the randomly selected points across the domain. We then choose a different set of sample test points across the domain using the same Latin hypercube sampling technique and pick a portion of the points that give the highest error in predicting f . We add this set of points to previous collocation points and train the network again. This procedure is important, as it prevents us from losing the accuracy of the solution across the entire domain and, in the meanwhile, helps us to focus more points to learn the trickier parts better. This process can be iterated a few times if necessary, by adding a different set of sampled collocation points to the original set and training again. An illustrative diagram is shown in Fig. 2.

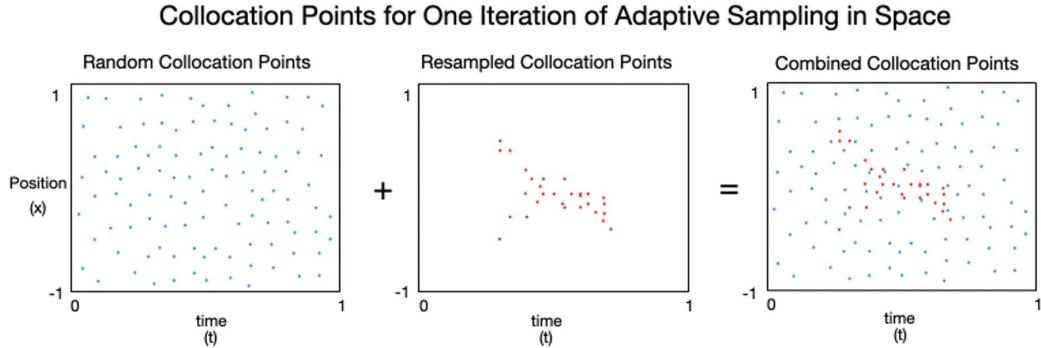


Figure 2: An illustration of adaptive sampling the collocation points over one iteration. Consider the domain $x \in [-1,1]$ and $t \in [0,1]$. The blue points show the set of randomly sampled collocation points using Latin hypercube sampling. Training on these points keeps the solution of the equation accurate across the whole domain. The red points show an example of a set of resampled collocation points sampled after evaluating the f -prediction network for the highest areas of error. These points help to improve the solution accuracy over red-point zones, which is usually the interfaces for phase field equations. The combined set of points is the collocation points used to train the network. The network can repeat this process for multiple re-sampling iterations. The blue points will stay the same, but the red points may change to focus on other parts of the domain that are not being learned well.

2.3.2 Adaptive strategies in time

The strategy of adaptive sampling collocation points addresses the moving interface for the phase field solutions. But for certain phase field problems with sharp transition, even the PINN with an adaptive sampling of collocation points fails to converge to the actual solution. Thus, some extra attention is still needed.

In this section, we introduce two adaptive strategies in time to improve the convergence of the PINN. The first time-adaptive approach is similar to the adaptive method introduced above, in that collocation points in time are strategically chosen to improve learning. The second time-adaptive method takes a different approach where we create separate networks on smaller (subsequent) time domains of fixed or adaptive length.

Time-adaptive approach I: adaptive sampling in time. At each time step of this approach, we require the data points, (initial, boundary, and general collocation both original and resampled) to come from within a specified time interval. For instance, if we are approximating the solution in the time domain $[0,1]$, we start with small time intervals $[0,t_1]$, $t_1 > 0$, where t_1 is close to zero, saying $t_1 = 0.1$. Then we gradually increase the time span, i.e., $[0,t_i]$, $i=1,2,\dots,N$, with $0 < t_1 < t_2 < \dots < t_N = 1$, when each time span is learned well. Eventually, the solution is learned well on the whole domain. This idea is illustrated in Fig. 3. In particular, one usually can set a threshold and a maximum training iterations. Once the loss function value is smaller than the threshold or the training exceeds the maximum training iterations, the training process marches onto next time step. In certain cases, if the loss function value is still huge after maximum training iterations, the time step size shall be reduced.

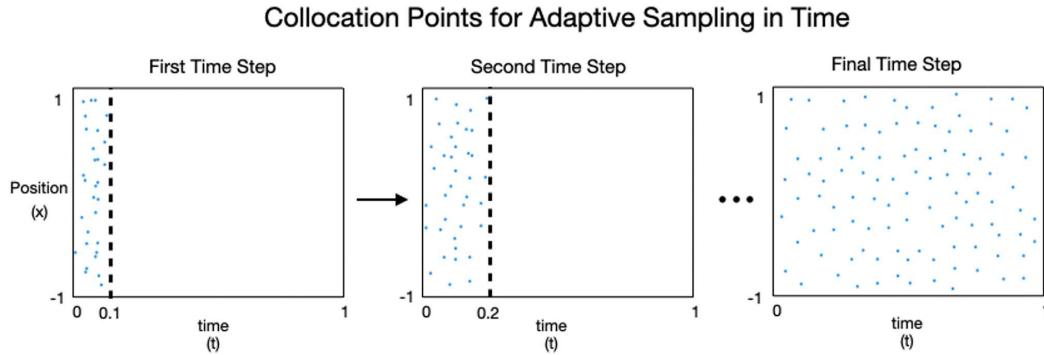


Figure 3: This figure illustrates the concept behind the time-adaptive approach I. Consider a time domain $[0,1]$. The first time step only allows data points to be taken between $t \in [0,0.1]$. Once the f -predicted error on the interval is sufficiently small, collocation points are then chosen on a larger interval $[0,0.2]$, including adaptive space sampling. Collocation points are still chosen from the earlier time domains to keep what has been learned there learned well. This is continued until the time interval covers the entire time range for the problem. Note this is all done on one PINN (in the next time method, multiple networks are created for each time interval).

In addition, this idea of adaptive sampling in time allows the user to designate a list of time steps. For each time step, collocation points will be sampled from only this restricted domain. Adaptive space sampling is used within the restricted time interval to improve learning. The network is then trained for each time interval using adaptive sampling collocation points. At each new iteration for the time step, the f -predicted error on that time domain is calculated.

Time-adaptive approach II: adaptive time marching strategy. In the second time-adaptive approach, we propose to split up the domain of interest into smaller problems. Notice that in the first time-adaptive approach, we only have a single network that focuses the collocation points adaptively in time. In the second approach, we create separate networks for each time step (interval). For example, if our domain of interest is $[0,1]$, and we set the time step $\Delta t = 0.1$, we train one network to learn the solution on the interval from $[0,0.1]$. Once the solution is learned well on this time interval, we train another network on the interval $[0.1,0.2]$, and so on, until we solve the problem in the entire time domain.

A caveat here is we cannot use the initial condition for the later networks as the given initial value is only valid for $t=0$. Instead, we use the prediction from the previous time step's network as the initial condition for the current network. We continue doing this until we have covered the whole time domain of the original problem. The individual networks can be combined to obtain a solution at any point in the domain of the original problem. The idea is illustrated in Fig. 4.

We remark that, for the solutions of phase field models, there are sharp transitions in both time and space, which make them very tough to solve. Utilizing the adaptive ideas (in both time and space), our improved (adaptive) PINNs improve the accuracy of the baseline PINN by avoiding certain local minima or saddle points.

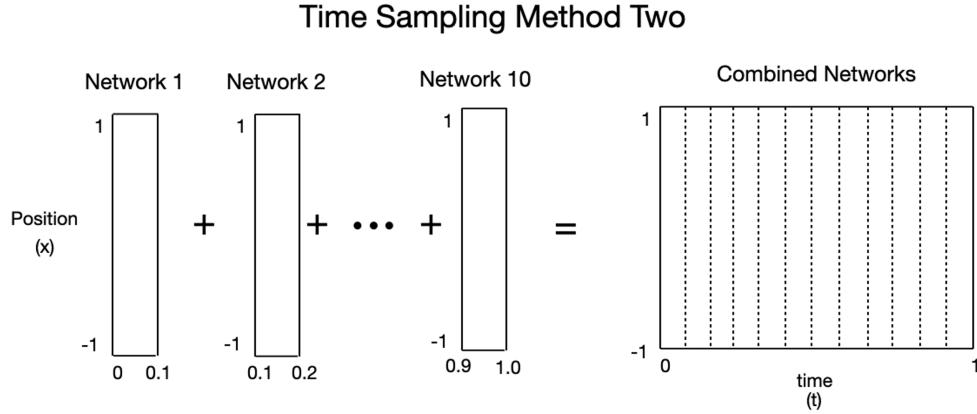


Figure 4: This figure illustrates the concept behind the adaptive time-marching strategy. Here we choose time step $\Delta t = 0.1$, and individual networks are trained for each time step. Network 2 shares the same initial time as the last time in Network 1. Once Network 1 has been learned well, its values at $t = 0.1$ can be used as the initial condition for Network 2. The solution for each individual network can be combined at the end into one continuous solution that covers the entire domain of the problem. Each network has the same time length but can be bigger or smaller for parts of the domain that are easier or harder to learn.

3 Numerical results

In this section, we will provide several numerical tests on solving the Allen-Cahn equation and the Cahn-Hilliard equation with the proposed strategies in the previous section.

First of all, let us recall the Allen-Cahn equation and Cahn-Hilliard equation. Denote the specific expression of the Ginzburg-Landau free energy

$$F = \int_{\omega} \frac{\gamma_1}{2} |\nabla u|^2 + \frac{\gamma_2}{4} (u^2 - 1) d\mathbf{x}, \quad (3.1)$$

where γ_1 and γ_2 are parameters. If we take the L^2 gradient flow, we obtain the Allen-Cahn equation

$$\partial_t u = \gamma_1 \Delta u + \gamma_2 (u - u^3); \quad (3.2)$$

and if we take the H^{-1} gradient flow, we obtain the Cahn-Hilliard equation

$$\partial_t u = \Delta (-\gamma_1 \Delta u + \gamma_2 (u^3 - u)). \quad (3.3)$$

Thermodynamically consistent boundary conditions (such as periodic boundary conditions and homogeneous Neumann boundary conditions) and initial values shall be proposed to close the system.

In the rest of this section, we use uniformly random samples from the initial and boundary points and use a Latin hypercube sampling (LHS) strategy to sample the collocation points in the interior domain. Recall the u -network gives the approximated value of the solution to the differential equation, and the f -network gives the approximated residual value of the equation at the given point, which should ideally be zero.

To test the accuracy of the learned solution, the ‘actual’ solution is obtained by solving the phase field equations using classical numerical methods with high accuracy. The accuracy of the trained model is assessed by taking the relative l_2 -norm of the difference between the ‘actual’ value $u(x_i, t_i)$ at those points and the u -network output $\mathcal{U}(x_i, t_i)$ at those points, i.e.,

$$Error = \frac{\sqrt{\sum_{i=1}^N |\mathcal{U}(x_i, t_i) - u(x_i, t_i)|^2}}{\sqrt{\sum_{i=1}^N |u(x_i, t_i)|^2}}, \quad (3.4)$$

given the data points $\{(x_i, t_i)\}_{i=1}^N$, with N the number of the points.

3.1 Solving the Allen-Cahn equation

We first tested the Allen-Cahn equation of (3.2) with periodic boundary conditions in one dimension, and chose the parameters: $\gamma_1 = 0.0001$ and $\gamma_2 = 5$. The specific system is summarized as follows:

$$\begin{aligned} u_t - 0.0001u_{xx} + 5u^3 - 5u &= 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ u(0, x) &= x^2 \cos(\pi x), \\ u(t, -1) &= u(t, 1), \\ u_x(t, -1) &= u_x(t, 1). \end{aligned} \quad (3.5)$$

Note that in [24], the authors did not test this problem with the (continuous) PINN. Instead, they solved it using the discrete Runge-Kutta neural network. We thus first test it with the PINN from [24] as a baseline.

As mentioned in previous section, we introduce two neural networks: the u -network as

$$\mathcal{U}: (x, t) \rightarrow \mathcal{U}(x, t), \quad (3.6)$$

and the f -network as

$$\mathcal{F}: (x, t) \rightarrow \mathcal{U}_t(x, t) - 0.0001\mathcal{U}_{xx}(x, t) + 5\mathcal{U}^3(x, t) - 5\mathcal{U}(x, t). \quad (3.7)$$

And the loss function is defined as

$$MSE = MSE_u + MSE_f + MSE_b, \quad (3.8)$$

where MSE_u and MSE_f are defined the same as (2.5) and the error due to boundary term is replaced by

$$MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(|\mathcal{U}(t_b^i, x_u) - \mathcal{U}(t_b^i, x_l)|^2 + |\mathcal{U}_x(t_b^i, x_u) - \mathcal{U}_x(t_b^i, x_l)|^2 \right) \quad (3.9)$$

to address the periodic boundary condition. Here N_b is the number of collocation points used on the boundary, $\{t_b^i\}_{i=1}^{N_b}$ are the time values for those points, x_u is the upper bound

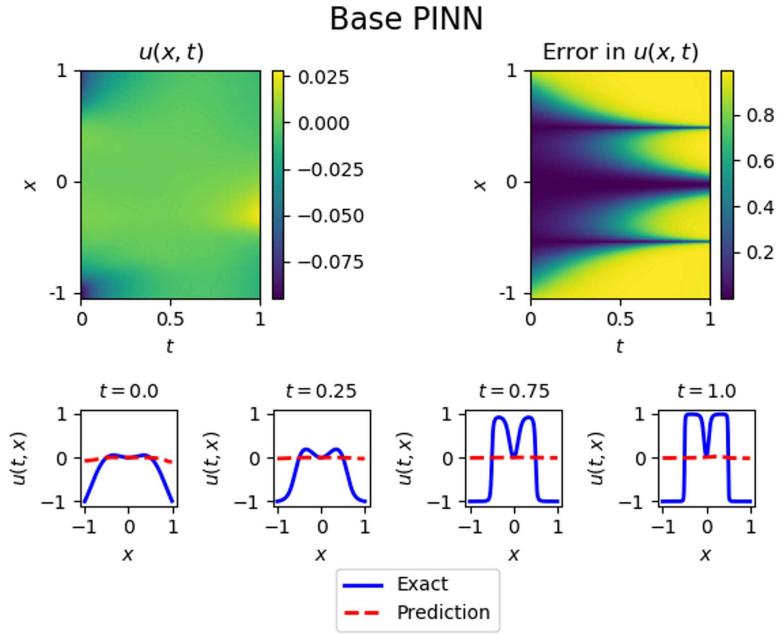


Figure 5: Solutions of the AC equation learned using the Base PINN method. The four plots on the bottom are the predicted solutions vs. accurate solutions at different times. The baseline PINN approach failed to learn the solution well.

for x , and x_l is the lower bound. For this problem we have $x_u = 1$ and $x_l = -1$ from our domain. We see that both the expressions inside the absolute values are ideally zero if the learned solution follows the boundary conditions set by the equation. In this problem (otherwise specified), we use $N_f = 10,000$ collocation points, $N_u = 200$ initial points, $N_b = 200$ boundary points, and tanh as the activation function. As a general practice in the rest of this paper, to optimize the loss function, we use Adam optimizer first and then use the L-BFGS-B optimizer (with respect to the whole training data) to fine-tune the neural network.

First of all, we tested the baseline PINN approach [24]. Unfortunately, using the baseline PINN approach alone, we were not able to learn the accurate solution for the Allen-Cahn equation. The relative l_2 error stayed around 0.99. The result is summarized in Fig. 5, and we observed that the predicted solution at different time steps is not close to the actual solution.

Then, we added the weights in the loss function, as proposed in Eq. (2.6). With this strategy, the neural network performed slightly better with a relative l_2 error of 0.52, but the algorithm still failed to converge, as shown in Fig. 6. The network learned the solution better at points at the starting time domain and near the boundary, but the learned solutions lost their accuracy in the later time domain, where the collocation points are sampled randomly.

Then, we used the mini-batching strategy introduced in Section 2.2.2. In this trial, we

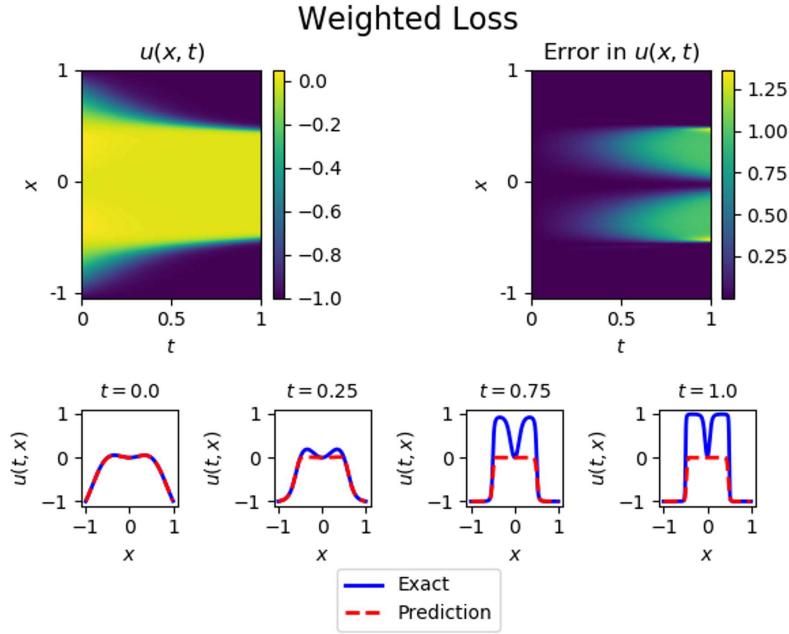


Figure 6: Solutions of the AC equation learned using the PINN method with weights in the loss function. Here we were still using the baseline PINN approach without changing the collocation point, but we added weights in the loss function by putting more weight on the initial condition vs. the collocation and boundary conditions. This results in a slight improvement, especially at times near $t=0$. The learned solution is still not accurate enough.

used 10,000 collocation points, 512 initial points, and 200 boundary points. This network had 4 hidden layers with 128 neurons per layer, and tanh as the activation function. The results are summarized in Fig. 7, where we chose mini-batch size as 32 with 100 epochs. The mini-batching trial performed almost as well as the sampling strategy in the previous trial. Notice, however, for both these solutions, the learned solutions near $t=1$ and $x=0$ were not exactly matching the actual solutions. By using time sampling approaches introduced in Section 2.3.2, the solutions could be learned accurately near both $t=1$ and $x=0$.

Next, we used the adaptive sampling approach introduced in Section 2.3.1. The network architecture was the same as the previous two trials. In this trial, we used 2000 original collocation points, along with 200 resampled collocation points for each iteration. With adaptive sampling collocation points, we obtained a much better solution with higher accuracy. After about six re-sampling iterations, the solution did not improve significantly. The final result is shown in Fig. 8. While the solution was much better, at the latter time steps, the learned solution didn't quite match the real solution. The relative l^2 error was improved from the others at $2.33e-02$.

The l_2 errors for various cases are summarized in Table 1. It is important to highlight also that this improved accuracy for the adaptive approach was obtained with only a

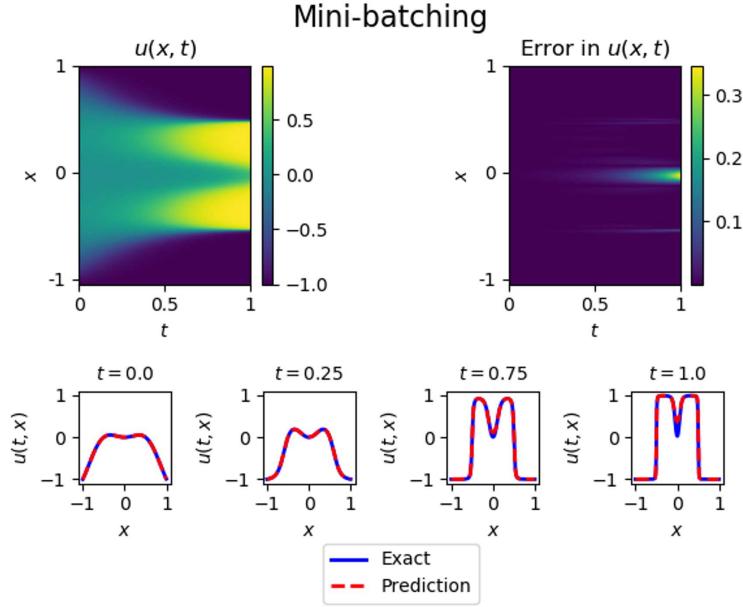


Figure 7: Solutions of the AC equation learned using the PINN method being trained with mini-batching approach. The mini-batching approach helped improve the accuracy more than adding weights in the loss function alone.

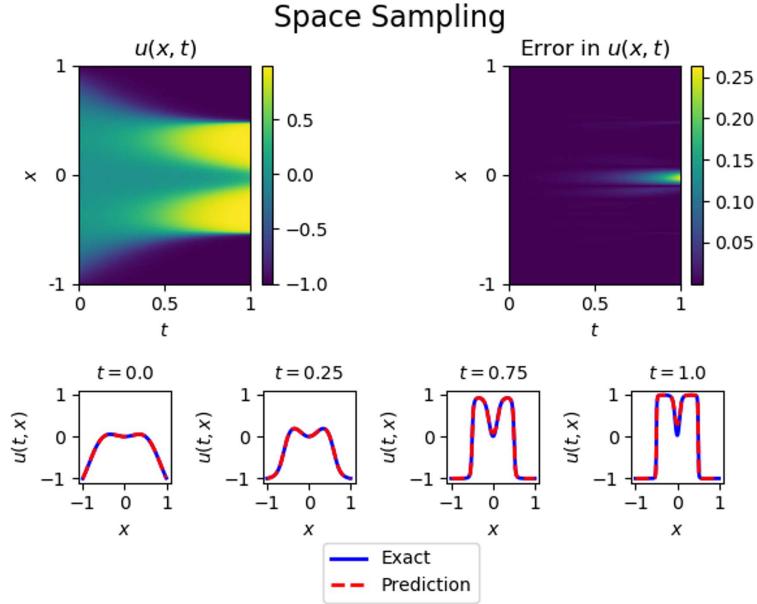


Figure 8: Solutions of the AC equation learned using the PINN method with weighted loss function and adaptive sampling collocation points during the iteration. This improved PINN approach learned an accurate solution for the AC equation. The focusing of sampling collocation points in the areas with less accuracy allowed the network to learn the solution better.

Table 1: Comparison of errors in the learned solutions of the Allen-Cahn equation using various PINN approaches: (1) the baseline PINN approach; (2) adding weights in the loss function; (3) adoptive resampling collocation points. The adaptive re-sampling approach produced the best result.

Allen-Cahn	PINN	Weighted Loss	Mini-batching	Re-sampling
Relative l_2	9.90e-1	5.22e-1	3.25e-2	2.33e-2
Relative l_1	9.90e-1	3.25e-1	8.80e-3	6.20e-3
l_∞ -norm	9.96e-1	1.37	3.37e-1	2.64e-1

fraction of the collocation points that the non-adaptive tests used. The adaptive test used only 2,000 collocation points while the others were tested with typical 10,000 collocation points. Thus, the adaptive approach noticeably reduces computational costs. We remark that the idea of adaptive re-sampling points is essential, as this same iteration scheme was tried without adding resampled points, and the solution did not converge as expected.

In the first example, we test the improved PINN method on solving the AC equation with a smooth interface. Next, we further investigate the capability of improved PINN on solving situations with sharper moving interfaces. In particular, we change the initial condition for the Allen-Cahn equation with more oscillations and use various values for the parameters to see how the methods work on different problems.

In the following series of tests, we used the AC equation as below

$$\begin{aligned}
 & u_t - \gamma_1 u_{xx} + \gamma_2 u^3 - \gamma_2 u = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\
 & u(0, x) = x^2 \sin(2\pi x), \\
 & u(t, -1) = u(t, 1), \\
 & u_x(t, -1) = u_x(t, 1).
 \end{aligned} \tag{3.10}$$

This equation is different from the one previously tested of (3.5) in the initial condition and that the gamma parameters are not set. Instead of testing one set of parameters, we varied them, in particular γ_2 , to see how our method performs on problems of increasing difficulty. We kept $\gamma_1 = 0.0001$ as in the previous problem.

We tested the adaptive re-sampling method using parameters $\gamma_2 = 1, 2, 3, 4$. Note when γ_2 increases, the transition interface of the solutions is sharper, which makes it harder to solve the AC equation numerically. As an agreement, we observed numerically that for smaller values of γ_2 , such as $\gamma_2 = 1$ and $\gamma_2 = 2$, the proposed resampling method in Section 2.3.1 converged in a reasonable number of re-sampling iterations. However, for $\gamma_2 = 3$, the resampling approach in Section 2.3.1 failed to converge within a reasonable number of re-sampling iterations. Note that this happened in fairly early time steps, before $t = .35$, and the error propagated and enlarged later on at time $t = 1$. It could be that a certain combination of the number of original collocation points with a number of resampled collocation points would help this solution to converge. However, we did not find one. This issue got even worse for $\gamma_2 = 4$. The neural network was unable to learn the larger

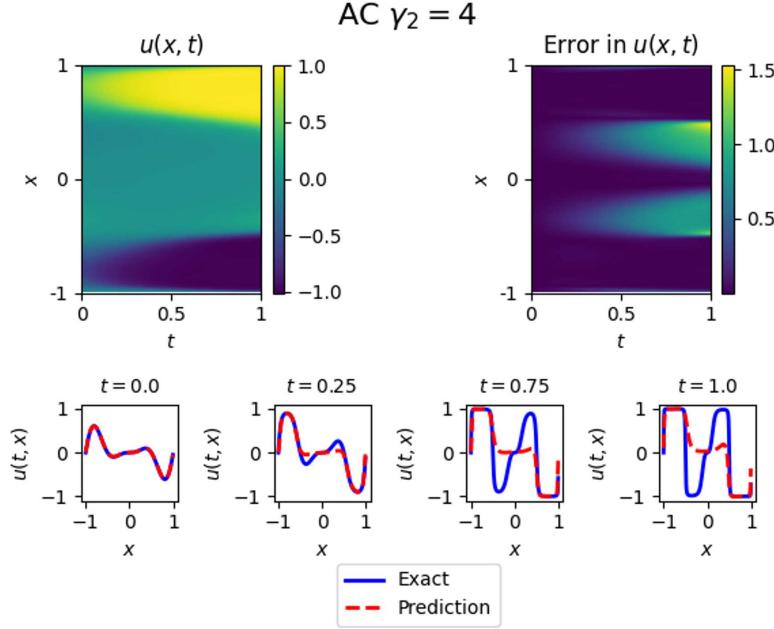


Figure 9: Solution results of the AC ($\gamma_2=4$). For this value of γ_2 , we see that the solution is not learned all the way even with adaptive sampling.

curves in this case. As seen in Fig. 9, the sharp transition layer was not well captured by the learned solution.

To overcome this, we utilized the adaptive in time sampling strategies introduced in Section 2.3.2. Finally, we saw the results for the trial using adaptive in time sampling approach I, as shown in Fig. 10. Using this approach, we were able to learn the solution for the more difficult problem with $\gamma_2=4$ much better with a relative l_2 error of 0.04. As seen in Fig. 10, the learned solutions are accurate enough across the domain. As has been observed in other tests, the sharp curve when time is close to 1 is not learned perfectly.

With time sampling approach II, we actually observed that difference to be smaller, as shown in Fig. 11. For the time sampling approach II on the Allen-Cahn equation, we saw that at times near $t=1$, the solution is learned better than it was in the past.

Next, we used the proposed time-adaptive approach to further solve some more complicated problems in higher dimensions. We focused on the classical benchmark problem: the shrinking of a single drop problem for the Allen-Cahn equation.

For the first benchmark problem, the Allen-Cahn model reads as

$$\partial_t \phi = \lambda(\varepsilon^2 \Delta \phi - \phi^3 + \phi), \quad \mathbf{X} \in \Omega, \quad t \geq 0, \quad (3.11)$$

We chose the domain $\Omega := [0, 1]^2$, the parameters $\lambda = 10$, $\varepsilon = 0.025$. the initial profile for ϕ

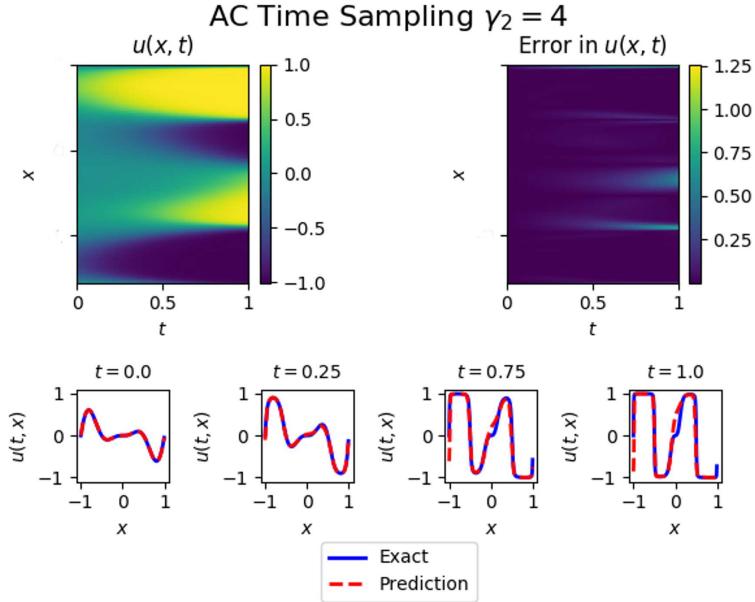


Figure 10: Solutions of the AC equation ($\gamma_2 = 4$) solved using the improved PINN with the time-adaptive approach I and adaptive sampling of collocation points. This equation could not be solved using any of the methods used in previous trials. With fixed time steps of length $\Delta t = 0.1$, this time-adaptive approach I focused on earlier times and then kept expanding the interval to encompass the whole domain.

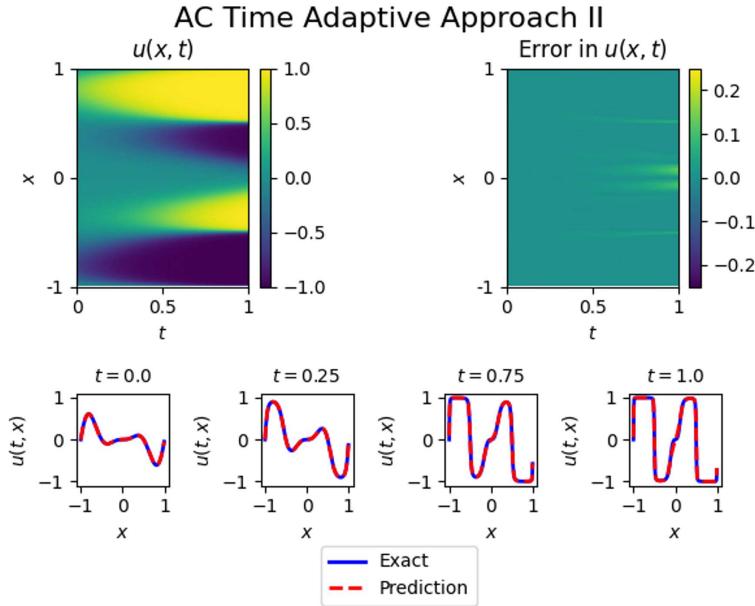


Figure 11: Solutions of the AC equation ($\gamma_2 = 4$) solved using the improved PINN with the time-adaptive approach II and mini-batch training. With fixed time steps of length $\Delta t = 0.25$, this time-adaptive approach II using multi-neural-networks to encompass the whole domain provides better accuracy.

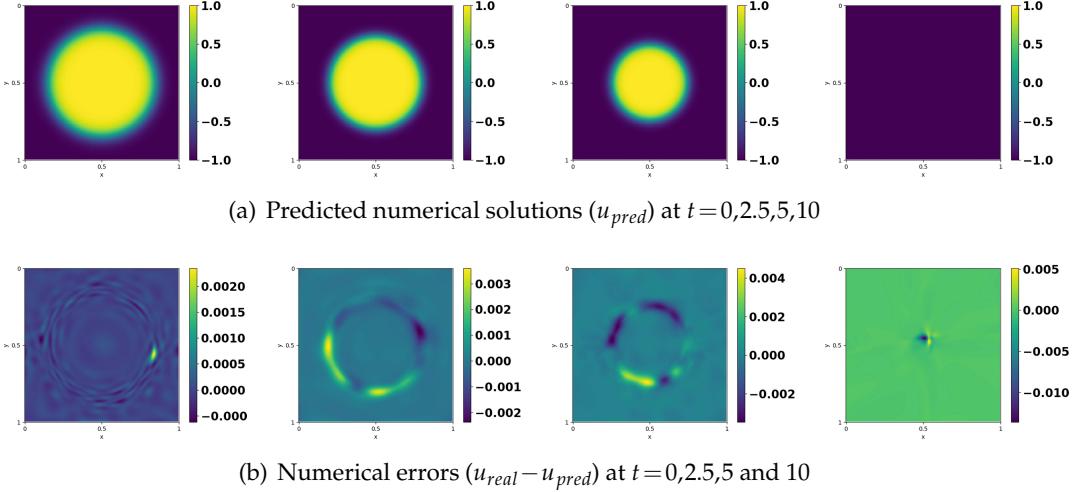


Figure 12: Numerical approximations of 2D Allen-Cahn benchmark problem using the time-adaptive approach II. In this example, we chose $\Delta t = 1$, and $C_0 = 10^3$ the neural network is $\mathcal{U}: (x, y, t) \rightarrow \mathcal{U}(x, y, t)$ with 6 hidden layers, 128 nodes per hidden layer. We chose time step $\Delta t = 0.2$. For each neural network, we used 100 epochs with batch size 32 for the Adam training, followed with a L-BFGS-B optimizer.

was given as

$$\phi(x, y, t=0) = \tanh\left(\frac{0.35 - \sqrt{(x-0.5)^2 + (y-0.5)^2}}{2\varepsilon}\right), \quad (3.12)$$

and we utilized the time-adaptive approach II to solve this problem for $t \in [0, 10]$ with fixed time step size $\Delta t = 1$. The predicted numerical solutions are shown in Fig. 12(a), where we observed the drop shrink and eventually disappeared. The numerical error, i.e. the difference between the real solution (which is computed accurate with classical numerical solver) and the predicted solutions of the neural network are shown in Fig. 12(b), where we observed the error was already very small. In other words, the time-adaptive approach II provided accurate approximation for this problem.

Similarly, the strategy can be used to solve the 3D benchmark problem as well. In this case, we chose $\Omega = [0, 1]^3$, with parameters $\lambda = 10$, $\varepsilon = 0.05$. the initial profile for ϕ was chosen as

$$\phi(x, y, z, t=0) = \tanh\left(\frac{0.35 - \sqrt{(x-0.5)^2 + (y-0.5)^2 + (z-0.5)^2}}{2\varepsilon}\right). \quad (3.13)$$

Then, the solutions for $t \in [0, 1]$ predicted by the time-adaptive approach II are summarized in Fig. 13, where we observed that the neural network provided accurate predictions.

Then, to further demonstrate the effectiveness of the time-adaptive approach II, we solved the Allen-Cahn equation in a complex geometry, an L shape domain defined as

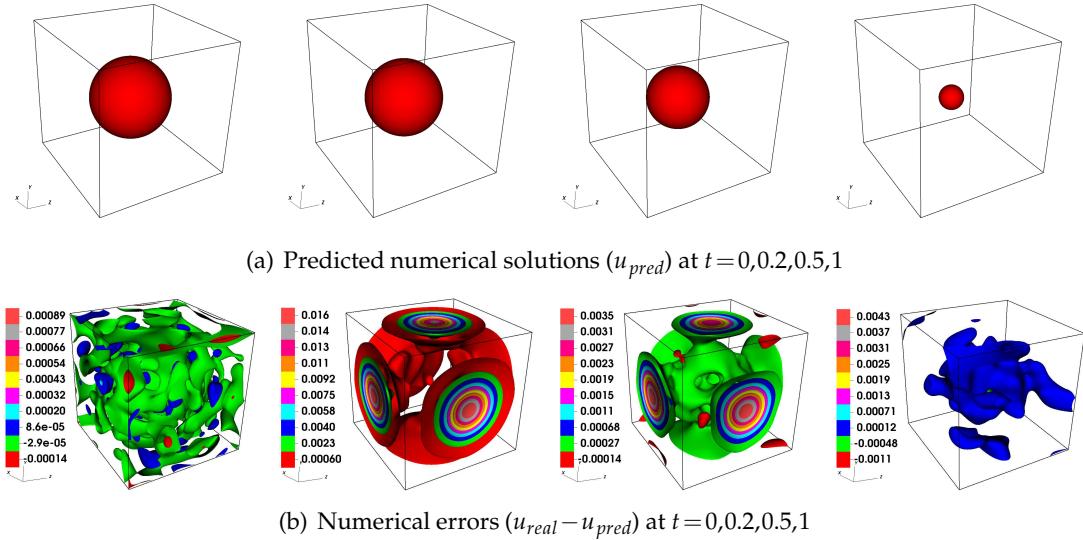


Figure 13: Numerical approximations of 3D Allen-Cahn benchmark problem using the time-adaptive approach II. The predicted numerical solutions and their numerical errors at various time steps are shown. In this example, we chose $\Delta t = 0.5$, and $C_0 = 10^3$ the neural network is $\mathcal{U}: (x, y, z, t) \rightarrow \mathcal{U}(x, y, z, t)$ with 6 hidden layers, 128 nodes per hidden layer. For each neural network, we used 100 epochs with batch size 32 for the Adam training, followed with a L-BFGS-B optimizer.

$\Omega := \{0 \leq x \leq 1, 0 \leq y \leq 0.5\} \cup \{0 \leq x \leq 0.5, 0.5 \leq y \leq 1\}$, with homogeneous Neumann boundary condition $\partial_n \phi|_{\partial\Omega} = 0$. And we chose $\lambda = 50$, $\varepsilon = 0.025$, along with the initial condition

$$\phi(x, y, t=0) = \tanh \frac{0.25 - \sqrt{(x-0.4)^2 + (y-0.4)^2}}{2\varepsilon}, \quad (x, y) \in \Omega. \quad (3.14)$$

The numerical results are shown in Fig. 14, where we observed the neural network predicted accurate dynamics of drop shrinking with relatively small errors.

3.2 Solving the Cahn-Hilliard equation

In the previous sub-section, we conducted a detailed study of the improved PINN on solving the Allen-Cahn equation. It indicated the proposed strategies have significantly improved the accuracy and convergence of the PINN.

Next we moved onto the Cahn-Hilliard Equation, which has higher order derivatives and is known to be harder to solve than the Allen-Cahn equation. Mainly, we focused on the following specific form

$$\begin{aligned} u_t - (\gamma_2(u^3 - u) - \gamma_1 u_{xx})_{xx} &= 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ u(0, x) &= -\cos(2\pi x), \\ u(t, -1) &= u(t, 1), \\ u_x(t, -1) &= u_x(t, 1), \end{aligned} \quad (3.15)$$

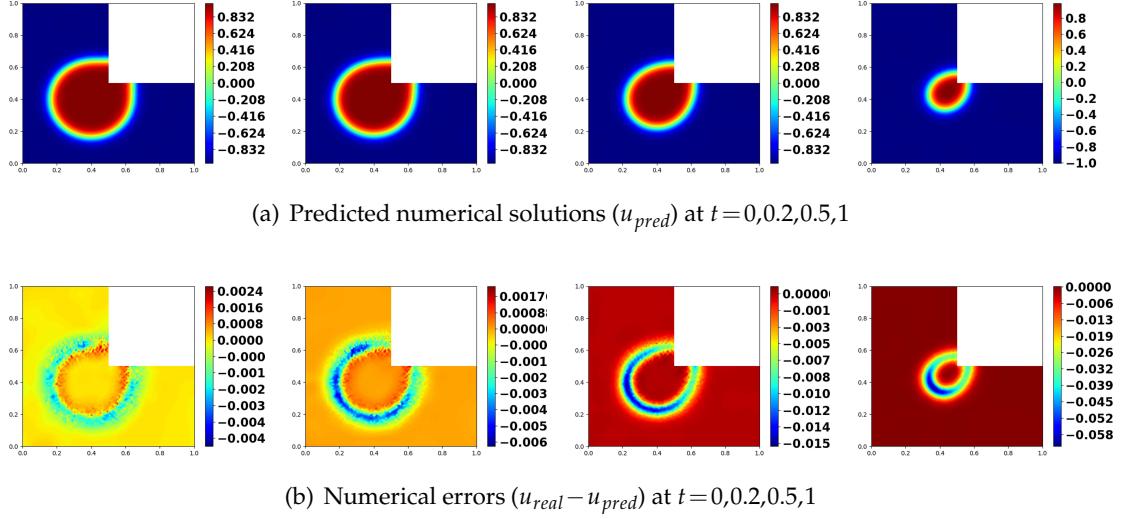


Figure 14: Numerical approximations of the Allen-Cahn equation in a L-shape domain using the time-adaptive approach II. In this example, the neural network is $\mathcal{U}: (x,y,z,t) \rightarrow \mathcal{U}(x,y,z,t)$ with 6 hidden layers, 128 nodes per hidden layer. We chose time step $\Delta t = 0.1$ and $C_0 = 10^3$. For each neural network, we used 100 epochs with batch size 32 for the Adam training, followed with a L-BFGS-B optimizer.

with γ_1, γ_2 the model parameters. For the following trials we used this equation with parameters fixed as $\gamma_2 = 0.01$, and $\gamma_1 = 10^{-6}$. We tested the same adaptive time method here as with Allen-Cahn with the regular adjustments to the f -network.

Notice the fact the back-propagation with high order derivatives are extremely expensive to calculate. To overcome this computational deficiency, we introduced an intermediate μ -network. Then, for solving the Cahn-Hilliard equation, we modified the networks as follows: the (u, μ) -network as

$$\mathcal{U}: (x,t) \rightarrow [\mathcal{U}(x,t), \mu(x,t)], \quad (3.16)$$

and the f -network as

$$\mathcal{F}: (x,t) \rightarrow \mathcal{U}_t(x,t) - \mu_{xx}. \quad (3.17)$$

And the loss function was defined as

$$MSE = MSE_u + MSE_b + MSE_f, \quad (3.18)$$

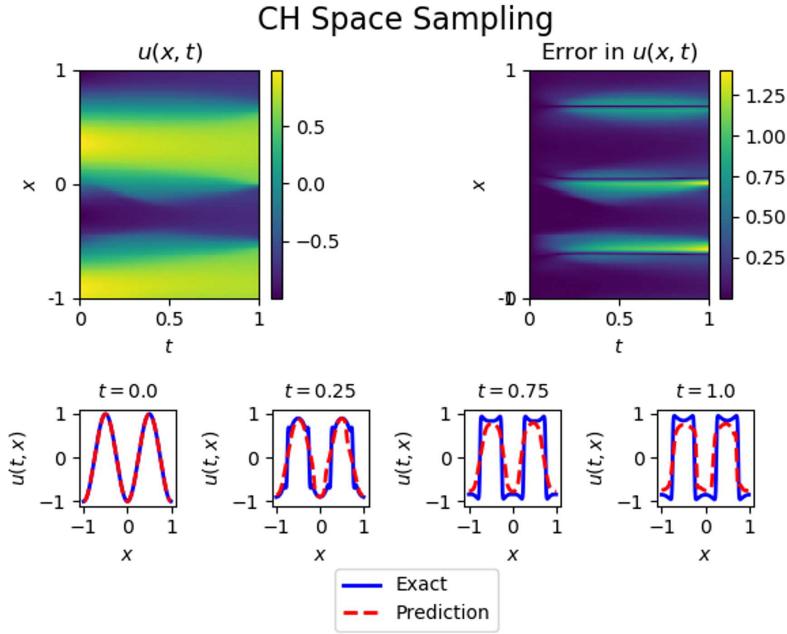


Figure 15: Solutions of the Cahn-Hilliard equation learned using the improved PINN with sampling collocation points alone. This figure indicates that training the PINN with sampling collocation points alone is not enough to achieve convergence for this problem even after many re-sampling iterations.

where the three terms were defined as

$$\begin{aligned}
 MSE_u &= \frac{1}{N_u} \sum_{i=1}^{N_u} |\mathcal{U}(0, x_u^i) - u^i|^2, \\
 MSE_b &= \frac{1}{N_b} \sum_{i=1}^{N_b} |\mathcal{U}(t_b^i, x_u^i) - \mathcal{U}(t_b^i, x_l^i)|^2 + |\mathcal{U}_x(t_b^i, x_u^i) - \mathcal{U}_x(t_b^i, x_l^i)|^2, \\
 MSE_f &= \frac{1}{N_f} \sum_{i=1}^{N_f} |\mathcal{F}(t_f^i, x_f^i)|^2 + |\mu(t_f^i, x_f^i) - \gamma_2(\mathcal{U}(t_f^i, x_f^i)^3 - \mathcal{U}(t_f^i, x_f^i)) + \gamma_1 \mathcal{U}(t_f^i, x_f^i)_{xx}|^2.
 \end{aligned} \tag{3.19}$$

This approach helps reducing the need to take the high order derivatives, and it turns out to significantly speed up computation and improve accuracy.

This Cahn-Hilliard equation in (3.15) was tested using the approaches of sampling collocation points in Section 2.3.1 and time sampling strategies in Section 2.3.2. We observed that the trial of sampling collocation points alone does not converge, as the results illustrated in Fig. 15. The neural network even failed to learn the solution well near $t=0$, and the error continued to accumulate throughout the entire domain. Like the trials for solving the AC equation, this trial used the same network architecture, weighted loss function, and the number of training data points. Each resampling iterations performed the Adam optimizer and the L-BFGS-B optimizer. The Adam optimizer learning rate is

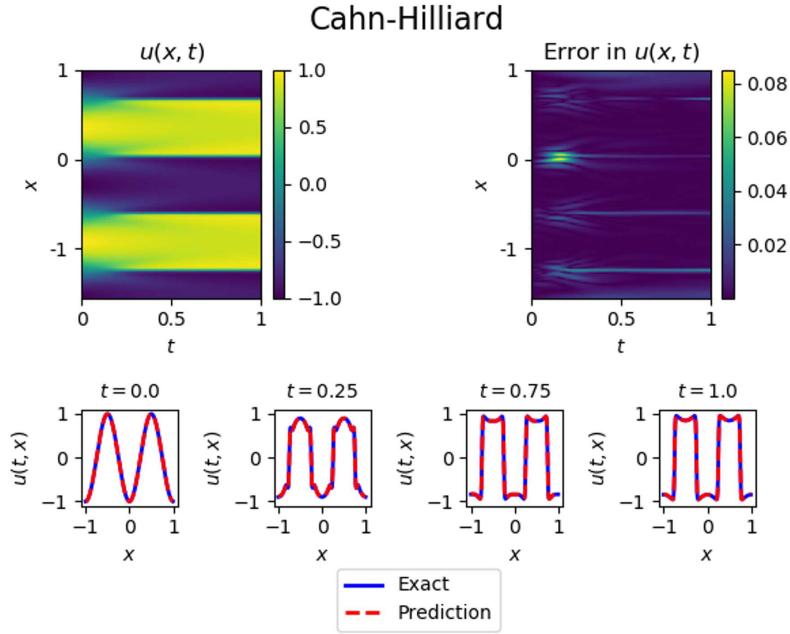


Figure 16: Solutions of the Cahn-Hilliard equation learned using the improved PINN with time-adaptive approach I in Section 2.3.2 and adaptive re-sampling collocation points in Section 2.3.1. The PINN with sampling collocation points alone was unable to learn the solution of the Cahn-Hilliard equation accurately. With the addition of adaptive time sampling, the improved PINN was able to learn an accurate solution with very small numerical errors.

set to 0.001, and the max iteration was set to 20,000. Even with 20 resampling iterations, the solution did not converge.

In the meanwhile, when using the time sampling approaches in Section 2.3.2, we observed noticeable improvement. The neural network was able to learn the solution well near $t=0$, and by gradually allowing collocation points to be sampled at later times, the neural network maintained this accuracy across the whole domain. Finally, we present the results for the trials run on solving the Cahn-Hilliard equation, which was the most difficult problem to solve thus far, in Fig. 16. From our observation, only methods that involved adaptive sampling in space and time were able to learn the solutions accurately. In Fig. 16, it shows the first trial using the time-adaptive approach I on the Cahn-Hilliard equation, we obtained our best result with a relative l_2 error of $9.51e-3$.

Next, we studied the benchmark problem for the Cahn-Hilliard model, which reads as

$$\partial_t \phi = \lambda \Delta (-\varepsilon^2 \Delta \phi + \phi^3 - \phi), \quad (x, y) \in \Omega, \quad t \geq 0. \quad (3.20)$$

In this example, we chose the domain $\Omega := [-1, 1]^2$, the parameters $\lambda = 1$, $\varepsilon = 0.05$, and the initial profile for ϕ as

$$\phi(x, y, t=0) = \max \left(\tanh \frac{r-R_1}{2\varepsilon}, \tanh \frac{r-R_2}{2\varepsilon} \right), \quad (3.21)$$

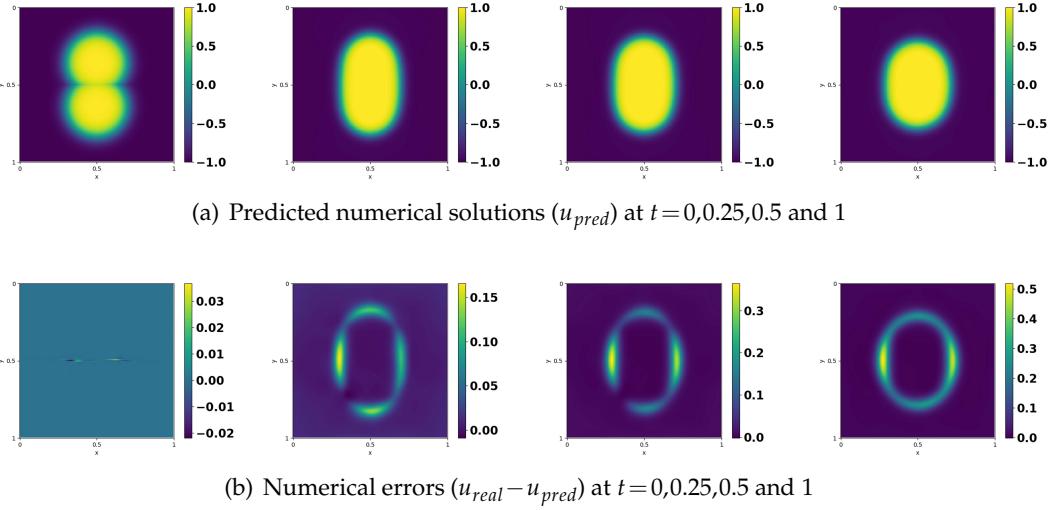


Figure 17: Numerical approximations of the 2D Cahn-Hilliard equation using the time-adaptive approach II. In this example, the neural network is $\mathcal{U}: (x,y,t) \rightarrow [\mathcal{U}(x,y,t), \mu(x,y,t)]$ with 6 hidden layers, 128 nodes per hidden layer. We chose time step $\Delta t = 0.2$ and $C_0 = 10^3$. For each neural network, we used 100 epochs with batch size 32 for the Adam training, followed with a L-BFGS-B optimizer.

where $r=0.4$, $R_1=\sqrt{(x-0.7r)^2+y^2}$, and $R_2=\sqrt{(x+0.7r)^2+y^2}$. This problem was solved for $t \in [0,1]$ by the time adaptive approach II, and the predicted solutions are summarized in Fig. 17(a) and the numerical errors (the difference between the real solutions and the predicted solutions) are summarized in Fig. 17(b). We observed the numerical solutions predicted by the adaptive approach II can capture the bubble merging accurately.

4 Conclusion

In this paper, we have introduced several strategies to improve the approximating capability of the physics informed neural network (PINN). Then we used the improved PINN to solve the phase field equations of increased complexity. Even though we focused on the problem of solving the Allen-Cahn equation and the Cahn-Hilliard equation, the improved PINN could readily be used to solve other difficult phase field equations as well.

Space sampling opened the door to other ideas of adaptive sampling. Space sampling uses the f -network predictions to pinpoint areas to focus data points on. Time sampling uses knowledge of differential equations to chose areas to focus on. We saw how both the value of the loss function and the values of the f -network predictions could help in determining if a network has learned a solution well. Using this information can help the network in making a decision, such as whether to use more collocation points or whether to focus on a smaller time domain. We also saw merit in both of the time sampling methods. The time-adaptive approach I proved better than just space sampling alone.

The time-adaptive approach II took a different angle that has the potential for even higher accuracy and learning the solution faster. It uses individual networks that can focus on a smaller problem domain. A potential downside with more difficult equations is that if the solution gets off on a time interval, all the intervals after that will propagate that error. It is important to learn the solution well on a time interval before moving on to the next one. This method may also be helpful when working on problems with larger time domains.

We have seen how adapting classical mathematical techniques and principles can help us find useful approaches for designing and training artificial deep neural networks. Roughly speaking, the best performance is obtained by using a combination of all of the techniques presented. More simple methods such as mini-batching, and adding weights in loss function are useful, especially when they are combined with more powerful adaptive sampling methods. This research has focused mainly on the problem of *solution* of differential equations. The next step of our future work is to test these approaches on the *discovery* of differential equations. The same neural network architectures can be used with the addition of a few more learnable/trainable parameters.

Acknowledgments

The authors would like to acknowledge the support from NSF-DMS-1816783 and NVIDIA Corporation for their donation of a Quadro P6000 GPU for conducting some of the numerical simulations in this paper.

References

- [1] C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk. Artificial neural network methods for the solution of second order boundary value problems. *Computers, Materials and Continua*, 59(1):345–359, 2019.
- [2] J. Berg and K. Nystrom. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.
- [3] J. Berg and K. Nystrom. Data-driven discovery of pdes in complex datasets. *Journal of Computational Physics*, 284:239–252, 2019.
- [4] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010*, pages 177–186. Springer, 2010.
- [5] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.
- [6] V. Buhrmester, D. Münch, and M. Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey. *arXiv*, page 1911.12116, 2019.
- [7] K. Cheng, C. Wang, and S. M. Wise. An energy stable BDF2 Fourier pseudo-spectral numerical scheme for the square phase field crystal equation. *Communications in Computational Physics*, 26:1335–1364, 2019.

- [8] J. M. Church, Z. Guo, P. K. Jimack, A. Madzvamuse, K. Promislow, B. Wetton, S. M. Wise, and F. Yang. High accuracy benchmark problems for Allen-Cahn and Cahn-Hilliard dynamics. *Communications in Computational Physics*, 26:947–972, 2019.
- [9] T. Dockhorn. A discussion on solving partial differential equations using neural networks. *arXiv*, page 1904.07200, 2019.
- [10] W. E and B. Yu. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6:1–12, 2018.
- [11] F. Guillen-Gonzalez and G. Tierra. Splitting schemes for a navier-stokes-cahn-hilliard model for two fluids with different densities. *Journal of Computational Mathematics*, 32(6):643–664, 2014.
- [12] D. Han and X. Wang. A second order in time uniquely solvable unconditionally stable numerical schemes for Cahn-Hilliard-Navier-Stokes equation. *Journal of Computational Physics*, 290(1):139–156, 2015.
- [13] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.
- [14] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.
- [15] C. Higham and D. Higham. Deep learning: An introduction for applied mathematicians. *arXiv*, page 1801.05894, 2018.
- [16] R. Kleinberg, Y. Li, and Y. Yuan. An alternative view: When does sgd escape local minima? *Proceedings of the 35th International Conference on Machine Learning*, 80:2698–2707, 2018.
- [17] B. Li, S. Tang, and H. Yu. Better approximations of high dimensional smooth functions by deep neural networks with rectified power units. *Communications in Computational Physics*, 27:379–411, 2020.
- [18] Y. Li, J. Kim, and N. Wang. An unconditionally energy-stable second-order time-accurate scheme for the cahn-hilliard equation on surfaces. *Commun Nonlinear Sci Numer Simulat*, 53:213–227, 2017.
- [19] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *arXiv*, page 1907.04502, 2019.
- [20] B. Mehlig. Artificial neural networks, 2019.
- [21] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv*, page 1811.03378, 2018.
- [22] M. Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19:1–24, 2018.
- [23] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv*, page 1801.01236, 2018.
- [24] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [25] P. Ramachandran, B. Zoph, and Q. V.Le. Searching for activation functions. *arXiv*, page 1710.05941, 2017.
- [26] S. Rudy, A. Alla, S. L. Brunton, and J. N. Kutz. Data-driven identification of parametric partial differential equations. *SIAM Journal on Applied Dynamical Systems*, 18(2):643–660, 2019.
- [27] S. Rudy, S. Brunton, J. Proctor, and J. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3, 09 2016.
- [28] J. Shen, C. Wang, X. Wang, and S. Wise. Second-order convex splitting schemes for gradient

- flows with Ehrlich-Schwoebel type energy: application to thin film epitaxy. *SIAM Journal of Numerical Analysis*, 50(1):105–125, 2012.
- [29] J. Shen and X. Yang. Numerical approximation of Allen-Cahn and Cahn-Hilliard equations. *Discrete and Continuous Dynamical Systems Series B*, 28(4):1669–1691, 2010.
 - [30] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.
 - [31] G. Wang, G. B. Giannakis, and J. Chen. Learning ReLU networks on linearly separable data: Algorithm, optimality, and generalization. *IEEE Transactions on Signal Processing*, 67(9):2357–2370, 2019.
 - [32] X. Yang and J. Zhao. On linear and unconditionally energy stable algorithms for variable mobility cahn-hilliard type equation with logarithmic flory-huggins potential. *Communications in Computational Physics*, 25:703–728, 2019.
 - [33] J. Zhao, Q. Wang, and X. Yang. Numerical approximations for a phase field dendritic crystal growth model based on the invariant energy quadratization approach. *Inter. J. Num. Meth. Engr.*, 110:279–300, 2017.