Adaptive Deep Learning Approximation for Allen-Cahn Equation [⋆]

Huiying Xu$^{1[0000-0002-3722-1048]}$, Jie Chen$^{1,*[0000-0003-1036-007X]}$, and Fei Ma$^{1[0000-0001-6099-480X]}$

Department of Applied Mathematics, School of Science, Xi'an Jiaotong-Liverpool University, Suzhou, 215123, China
jie.chen01@xjtlu.edu.cn
https://www.xjtlu.edu.cn/en/study/departments/academic-departments/applied-mathematics/

Abstract. Solving general non-linear partial differential equations (PDE) precisely and efficiently has been a long-lasting challenge in the field of scientific computing. Based on the deep learning framework for solving non-linear PDEs *physics-informed neural networks* (PINN), we introduce an adaptive collocation strategy into the PINN method to improve the effectiveness and robustness of this algorithm when selecting the initial data to be trained. Instead of merely training the neural network once, multi-step discrete time models are considered when predicting the long time behaviour of solutions of the Allen-Cahn equation. Numerical results concerning solutions of the Allen-Cahn equation are presented, which demonstrate that this approach can improve the robustness of original neural networks approximation.

Keywords: Deep learning · Adaptive collocation · Discrete time models · Physics informed neural networks · Allen-Cahn equation.

1 Introduction

Non-linear partial differential equations (PDE) play an important role in numerous research areas including engineering, physics and finance. However, solving general non-linear PDEs precisely and efficiently has been a long-lasting challenge in the area of scientific computing [4,6,11,13,15]. Numerical methods such as finite difference and finite element methods (FEM) are popular approaches in solving PDEs, due to their capability in solving non-linear problems and great freedom they can provide in the choice of dispersion. These methods discretise continuous PDEs and create simple basis functions on the discretised domain Ω. Through solving the system of basis coefficients, they approximate the true solutions of the targeted problems. Notwithstanding significant advance of these methods achieved in the past few decades and their capability to deal with fairly

[⋆] This work is partially supported by Key Program Special Fund in XJTLU (KSF-E-50, KSF-E-21) and XJTLU Research Development Funding (RDF-19-01-15).

complicated and oscillating problems, they consume a lot of time and computing resources, especially in the analysis and computation of complex problems.

Confronted with the shortcoming of traditional numerical methods in solving PDEs, researchers turned to deep learning techniques to simulate true results. Deep learning, a subset of machine learning containing powerful techniques that enable computers to learn from data, has attracted enormous interest among people in various fields. Deep learning is a great breakthrough of traditional machine learning algorithms on account of complicated non-linear combinations of input and output features using multiple hidden layers. During training process, the back-propagation algorithm increases weights of the combinations useful to obtain final output features, while gradually deprecate useless internal relations, making the approximation outcomes more accurate step by step. Great efforts have been made to obtain solutions of PDEs based on deep learning approaches. Neural networks are used to improve the precision of results obtained from finite difference method [8]. In [14], the authors apply regression analysis technique to develop a discretisation scheme based on neural networks in the solution of non-linear differential equations. To handle general non-linear PDEs, physics-informed neural networks (PINN) are trained to infer solutions, and the obtained surrogate models are differentiable in regard to all parameters and input data [11]. Deep Galerkin Method (DGM), a similar approach to PINN, is proposed to solve PDEs of high dimension through training a deep neural network to approach the differential operation, boundary conditions and initial condition [13]. Exploring the variational form of PDEs is another attrative approach to researchers [6]. Deep Ritz method (DRM) is one typical example of this approach, which casts governing equations in an energy-minimisation framework to solve PDEs [6].

The rest of this paper is organized as follows. In Section 2, we first briefly introduce artificial neural networks, and then provide a detailed explanation to PINNs. The idea of adaptive collocation is also introduced in this section. Using the examples of Allen-Cahn equation, we discuss how to simulate the results with the discrete time models of PINNs in Section 3, covering the structure of neural networks employed and the working mechanism. Adaptive collocation strategy is applied in selecting training points at initial and later stages when training PINNs. Several approximation outcomes are demonstrated, which are later compared to the results obtained from the algorithm without adaptive strategy. A conclusion and remark is presented in Section 4.

2 Methodology

2.1 Artificial Neural Networks

Neural networks are composed of three kinds of layers – input layer, hidden layers and output layer. The model with more than one hidden layer is referred to as deep neural network (DNN). Figure 1 displays the structure of a DNN with n hidden layers. The realisation of neural networks relies on two algorithms – feed-forward propagation and back-propagation. The former builds the framework for neural networks, while the latter optimises the constructed model to achieve desired results.

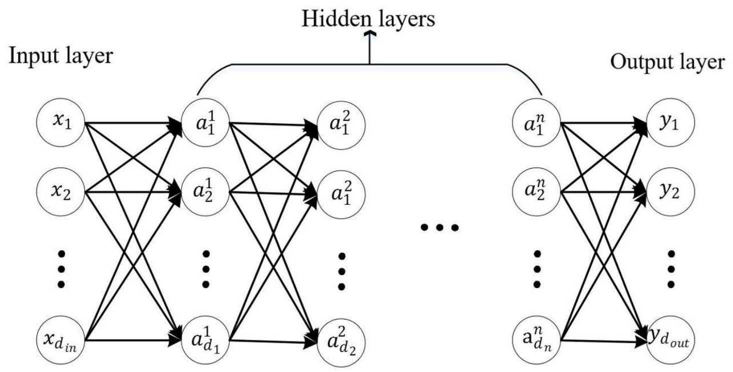

Fig. 1. Representation of a DNN as a graph. The number of nodes in the input layer and the output layer depends on the problem setting, so should be unchangeable, whereas the number of layers and nodes in each hidden layer could be modified to optimise the structure.

Feed-forward propagation algorithm In a *feed-forward neural network* where the input $\mathbf{x}$ contains d_{in} nodes, the output $\mathbf{y}$ contains d_{out} nodes, and the i^{th} hidden layer $\mathbf{a}_i$ consists of d_i nodes with $i = 1, ..., n$:

$$\begin{aligned}\mathbf{a}_1 &= f(\mathbf{x}; \mathbf{w}_1, b_1) = \mathbf{x}^\top \mathbf{w}_1 + b_1,\\ \mathbf{a}_i &= f(\mathbf{a}_{i-1}; \mathbf{w}_i, b_i) = \mathbf{a}_{i-1}^\top \mathbf{w}_i + b_i,\\ \mathbf{y} &= f(\mathbf{a}_n; \mathbf{w}_{n+1}, b_{n+1}) = \mathbf{a}_n^\top \mathbf{w}_{n+1} + b_{n+1},\end{aligned} \tag{1}$$

where $i = 2, 3, ..., n$, $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{n+1}$ are the weights and $b_1, b_2, ..., b_{n+1}$ are the biases. Thus, combining the equations in Eq. (1) we get the the prediction for output value $\mathbf{y}$:

$$\mathbf{y} = f(f(\cdots f(\mathbf{x})\cdots)). \tag{2}$$

However, if the neural network is designed in this way, combining all these hidden layers will work the same as merely one hidden layer. Based on this concern, the linear function f here is transformed to a non-linear one, denoted by h known as an activation function. Sigmoid units, tanh units and rectified linear units (ReLU) are the three most popular ones, while ReLU as the default activation function is conventionally used in hidden layers [9]. ReLU is defined as follows:

$$h(x) = \max(0, x). \tag{3}$$

Since ReLU are quite close to linear functions, they are able to preserve more properties, facilitating the optimisation. Hence, in the renewed feed-forward neural network, the output value $\mathbf{y}$ can be obtained by the following:

$$\begin{aligned}
\mathbf{a}_1 &= h(f(\mathbf{x}; \mathbf{w}_1, b_1)) = h(\mathbf{x}^\top \mathbf{w}_1 + b_1), \\
\mathbf{a}_i &= h(f(\mathbf{a}_{i-1}; \mathbf{w}_i, b_i)) = h(\mathbf{a}_{i-1}^\top \mathbf{w}_i + b_i), \\
\mathbf{y} &= h(f(\mathbf{a}_n; \mathbf{w}_{n+1}, b_{n+1})) = h(\mathbf{a}_n^\top \mathbf{w}_{n+1} + b_{n+1}).
\end{aligned} \tag{4}$$

Back-propagation algorithm After the framework for feed-forward neural network has been constructed, it comes to the training process to optimise the prediction of the output value $\mathbf{y}$, known as *back-propagation*. The most fundamental algorithm in back-propagation is *Gradient Descent*. In this part, the weights $\mathbf{w}$ and the biases b are trained through minimising the *cost function* J:

$$J(\mathbf{w}, b) = \frac{1}{d_{out}} \sum_{j=1}^{d_{out}} L(\hat{y}^{(j)}, y^{(j)}), \tag{5}$$

where y is the true value whereas $\hat{y}$ denotes the prediction value acquired from Eq. (4). The choice of function L depends on the problem to be solved. For example, cross-entropy loss function or exponential loss function is more suitable to be applied in binary classification problem. In regression problems, it is more appropriate to use L_2 loss defined as follows:

$$J(\mathbf{w}, b) = \frac{1}{d_{out}} \sum_{j=1}^{d_{out}} (\hat{y}^{(j)} - y^{(j)})^2. \tag{6}$$

Now, the problem is simplified to find:

$$(\mathbf{w}, b) = \arg\min_{\mathbf{w}, b} J(\mathbf{w}, b). \tag{7}$$

In Gradient Descent algorithm, the weights $\mathbf{w}$ and the biases b are trained following the formulas:

$$\mathbf{w} := w - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}}, \qquad b := b - \alpha \frac{\partial J(\mathbf{w}, b)}{\partial b} \tag{8}$$

where α is the *learning rate*. It is a crucial hyperparameter deciding the rate of convergence. Apart from gradient descent algorithm, a few advanced optimisers can be used to optimise the training process, including Momentum, Adagrad, RMSprop and Adam.

2.2 Physics-Informed Neural Networks

In this work, we consider parametrised non-linear PDEs of the following general form:

$$u_t + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, T], \tag{9}$$

where $u(t;x)$ is the latent solution, $\mathcal{N}[\cdot]$ represents a non-linear operator, and Ω is the domain in $\mathbb{R}^D$. Without the requirement to consider linearisation, prior assumptions, or division of local time interval, we can directly handle the non-linear problem in this setup. Additionaly, a large variety of problems in mathematical physics are encapsulated in Eq. (9), including kinetic equations, diffusion processes, conservation laws, and so on [11].

Generally, we use the discrete time models of PINNs, in which q-stage Runge-Kutta methods [10] are applied to Eq. (9):

$$\begin{aligned} u(t_0 + c_j \Delta t, x) &= u(t_0, x) - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u(t_0 + c_j \Delta t, x)],\ i = 1, \cdots, q, \\ u(t_0 + \Delta t, x) &= u(t_0, x) - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u(t_0 + \Delta t, x)]. \end{aligned} \tag{10}$$

Here, Δt is a predetermined value and $[t_0, t_0 + \Delta t]$ refers to the short time interval we are supposed to study on. The above equations depend on the parameters $\{a_{ij}, b_j, c_j\}$ which are fully determined by Runge-Kutta methods. To simplify the formulas, express Eq. (10) as

$$\begin{aligned} u_i^0(x) &= u(t_0 + c_j \Delta t, x) + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u(t_0 + c_j \Delta t, x)],\ i = 1, \cdots, q, \\ u_{q+1}^0(x) &= u(t_0 + \Delta t, x) + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u(t_0 + \Delta t, x)]. \end{aligned} \tag{11}$$

Thus, $u_i^0(x) = u_{q+1}^0(x) = u(t_0, x)$ for $i = 1, \cdots, q$. Taking x as the input feature and the following

$$[u_1^0(x), u_2^0(x), \cdots, u_q^0(x), u_{q+1}^0(x)] \tag{12}$$

as output features, together with the setting of several hidden layers, we have constructed the basic structure of PINNs. Each node of the output features $u_i^0(x)$ for $i = 1, \cdots, q$ and $u_{q+1}^0(x)$ are all equivalent to $u(t_0, x)$, i.e., the true value of u at the initial stage. Substituting $u(t_0, x)$ by what is obtained from the neural network in Eq. (10), we can predict the values along the short time interval, i.e., $u(t_0 + c_j \Delta t, x)$ for $i = 1, \cdots, q$ and $u(t_0 + \Delta t, x)$.

After determining the input and output features of the neural network, we enable the algorithm to approach to the true solution through minimising the cost function J defined as follows:

$$J(\mathbf{w}, b) = J_u + J_b, \tag{13}$$

where J_u refers to the difference between the approximation results $u_1^0(x)$, $\cdots$, $u_q^0(x)$, $u_{q+1}^0(x)$ and the true solution $u(t_0, x)$ of the PDE at initial stage, and J_b is the deviation of the prediction from the true value at boundary points at $t = t_0 + c_1 \Delta t, \cdots, t_0 + c_q \Delta t, t_0 + \Delta t$. Both J_u and J_b use mean squared error to reflect the variation.

2.3 Adaptive Collocation Strategy

Various methods of adaptive collocation can be employed to select training data of the neural networks. For instance, the authors use uniformly distributed training points at the initial stages when solving PDEs based on deep learning techniques, while add more points according to the residual values at later stages [2, 3, 7]. In [11], when training the discrete time algorithms of PINNs, certain number of random training points are generated from the domain Ω at the initial stage. In the case of short-time prediction, instead of choosing training points randomly, we propose to apply one approach of adaptive collocation strategy, and select points based on the real solution of the PDE. However, for the prediction of long-term solution, single-step of the discrete time algorithms may be inapplicable. In this case, we feed last estimation results of the model into the neural network as the sample data, and select training points from it adaptively.

Figure 2 illustrates the idea of our adaptive collocation strategy. The curve is assumed to be the true solution of the PDE at the initial stage, and the blue dots represent the selected training data. In the given simple example, both point-selection strategies choose 16 points from the domain, but the left picture sub-samples training data randomly, while the right one adaptively locates the points according to the value of u. Points have larger possibility to be selected around $u = 0$.

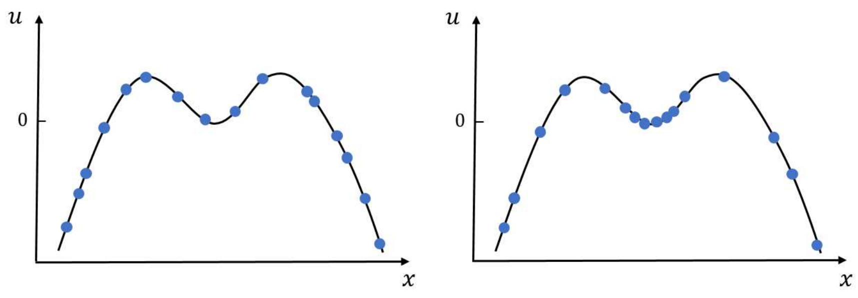

Fig. 2. Simple examples of sub-sampling training data. The left picture opts data randomly, while the right one uses adaptive collocation.

3 Numerial Experiments

3.1 Allen-Cahn Equation

Aiming to understate the capability of proposed discrete time models when handling non-linear PDEs, we take the time-fractional Allen-Cahn equation with period boundary conditions (14) as an example, which is a classical phase-field model.

$$\begin{aligned} &u_t - 10^{-4}u_{xx} + 5u^3 - 5u = 0,\ x\in[-1,1],\ t\in[0,2],\\ &u(0,x) = x^2\cos(\pi x),\\ &u(t,-1) = u(t,1),\\ &u_x(t,-1) = u_x(t,1). \end{aligned} \tag{14}$$

Originally, the Allen-Cahn equation was introduced to demonstrate the motion of a curved anti-phase boundary on multi-component alloy systems [1]. Through a phase-field approach, the Allen-Cahn equation has found its application in various research areas including fluid dynamics and complex moving interface problems [12]. For the Allen-Cahn equation (14), the non-linear operator in Eq. (9) is given by:

$$\mathcal{N}[u] = -10^{-4}u_{xx} + 5u^3 - 5u, \tag{15}$$

and $\mathcal{N}[u]$ is expressed as follows in the discrete form:

$$\begin{aligned} \mathcal{N}[u(t_0 + c_j\Delta t, x)] &= -10^{-4}u_{xx}(t_0 + c_j\Delta t, x) + 5[u(t_0 + c_j\Delta t, x)]^3\\ &\quad - 5u(t_0 + c_j\Delta t, x),\ j = 1, \cdots, q,\\ \mathcal{N}[u(t_0 + \Delta t, x)] &= -10^{-4}u_{xx}(t_0 + \Delta t, x) + 5[u(t_0 + \Delta t, x)]^3\\ &\quad - 5u(t_0 + \Delta t, x). \end{aligned} \tag{16}$$

In the neural network that calculate solutions for the Allen-Cahn equation (14), the cost function J is defined as:

$$J(\mathbf{w}, b) = J_u + J_b,$$

where

$$J_u = \sum_{i=1}^{q+1} |u_i^0(x) - u(t_0, x)|^2, \tag{17}$$

and

$$\begin{aligned} J_b = &\sum_{j=1}^{q} |u(t_0 + c_j\Delta t, -1) - |u(t_0 + c_j\Delta t, 1)|^2\\ &+ |u(t_0 + \Delta t, -1) - |u(t_0 + \Delta t, 1)|^2\\ &+ \sum_{j=1}^{q} |u_x(t_0 + c_j\Delta t, -1) - |u_x(t_0 + c_j\Delta t, 1)|^2\\ &+ |u_x(t_0 + \Delta t, -1) - |u_x(t_0 + \Delta t, 1)|^2. \end{aligned} \tag{18}$$

Sample data are generated through simulating the Allen-Cahn equation (14) using the Chebfun package [5], which employs conventional spectral methods. An explicit Runge-Kutta integrator with step $\Delta t = 10^{-5}$ is used. Along the phase we study on, solutions of 2048 evenly distributed points in the domain $\Omega = [-1, 1]$ are computed, forming the training and test data set.

3.2 Short-time Prediction

In this subsection, we merely look on the prediction of solutions of the Allen-Cahn equation (14) from $t_0 = 0.1$ to $t_1 = 0.9$, i.e., $\Delta t = 0.8$. Figure 3 shows the simulation outcomes using the Chebfun package. Firstly, 200 training data

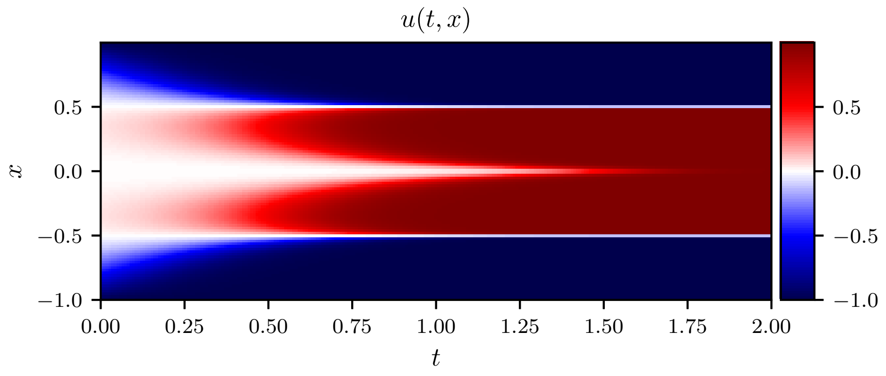

Fig. 3. Simulation results using conventional spectral models for $t \in [0,2]$.

are sub-sampled randomly from the 2048 points at the initial stage of $t_0 = 0.1$. To predict the solution of the equation at $t_1 = 0.9$, the discrete time models of PINNs are used with 4 hidden layers consisting of 200 nodes each. The output layer contains 101 neurons of $u(t_0 + c_j \Delta t, x)$ for $j = 1, \ldots, 100$ and $u(t_0 + \Delta t, x)$ corresponding to 100 stages of Runge-Kutta method. In Figure 4, the location of sub-sampled initial training data and the prediction results at the final stage are displayed. The small relative error reflects effectiveness of current model. However, we wonder whether selecting points adaptively corresponding to the value of u at $t = 0.1$ could further improve the accuracy. To this end, among the sample input data $\{x_1, \cdots, x_{2048}\}$, x_k has bigger possibility to be selected when $u(t_0, x_k)$ is closer to 0 for $k = 1, \cdots, 2048$. The following formula shows the method we use to calculate the possibility of x_k to be selected. Denote this possibility by $\mathbb{P}(x_k)$.

$$\begin{aligned} & f(x_k) = g(1 - |u(t_0, x_k)|) + \varepsilon, \ k = 1, \cdots, 2048, \\ & where\ g(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}, \\ & \mathbb{P}(x_k) = \frac{f(x_k)}{\sum_{m=1}^{2048} x_m}. \end{aligned} \tag{19}$$

Similarly, the location of selected initial training data and the prediction results at the final stage are shown in Figure 5. Indeed, accuracy of the estimated solution at $t = 0.9$ greatly improves compared to random data-selection approach.

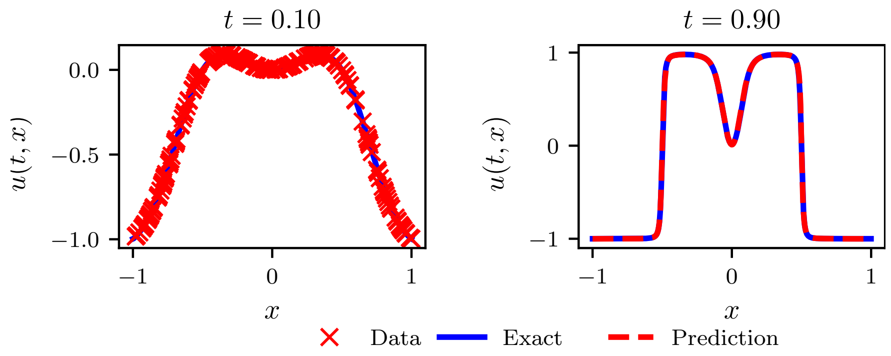

Fig. 4. The left picture depicts the 200 initial training points randomly sub-sampled from the data set of 2048 points, while the right one shows final prediction at $t_1 = 0.9$. The mean relative error on the whole data set is $1.013{\cdot}10^{-2}$.

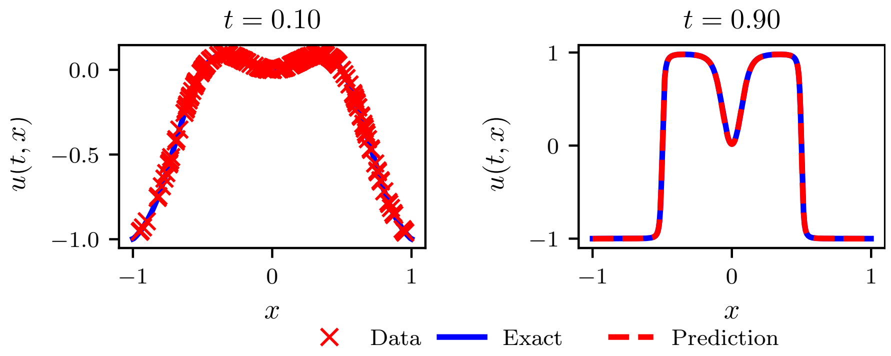

Fig. 5. The left picture depicts the 200 initial training points adaptively selected from the data set of 2048 points based on the value of $u(t_0, x_k)$ for $k = 1, \cdots, 2048$, while the right one shows final prediction at $t_1 = 0.9$. The mean relative error on the whole data set is $1.415{\cdot}10^{-3}$.

3.3 Long-time Prediction

In Section 3.2, single-step discrete time model of PINNs is employed to predict solutions of the Allen-Cahn equation (14) from $t_0 = 0.1$ to $t_1 = 0.9$. In this part, we consider to expand the interval to $[0.1, 1.5]$. Figure 6 displays the location of randomly selected initial training data and the prediction of u at $t_2 = 1.5$ using the same model as in Section 3.2. Applying adaptive collocation strategy described in Eq. (19), approximated solution at t_2 is shown in Figure 7. Easy

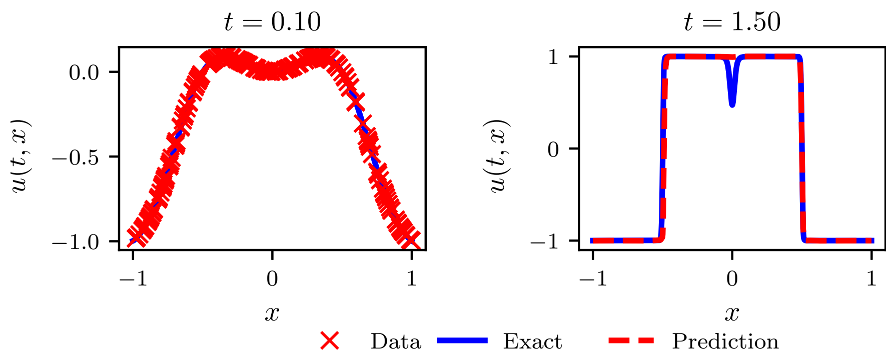

Fig. 6. The left picture depicts the 200 initial training points randomly sub-sampled from the data set of 2048 points, while the right one shows final prediction at $t_2 = 1.5$. The mean relative error on the whole data set is 0.1123.

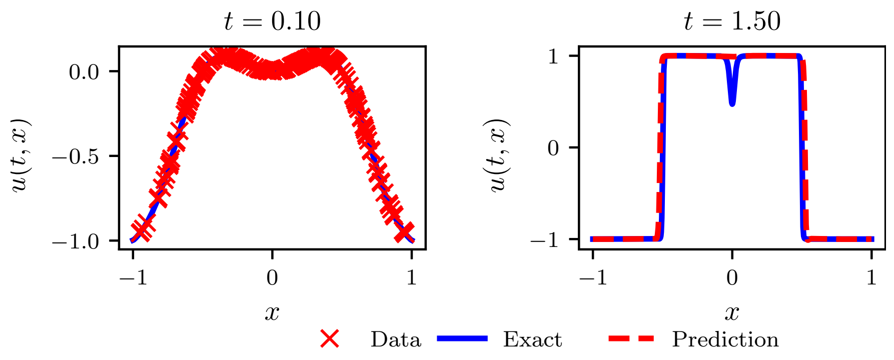

Fig. 7. The left picture depicts the 200 initial training points adaptively selected from the data set of 2048 points based on the value of $u(t_0, x_k)$ for $k = 1, \cdots, 2048$, while the right one shows final prediction at $t_2 = 1.5$. The mean relative error on the whole data set is 0.2447.

to find from the figures that one-step discrete time models fed with randomly or adaptively generated training data does not work satisfactorily. In the case of long-time prediction, adaptive collocation strategy loses its effectiveness.

Given the undesirable approximation results, we employ multi-step discrete time models of PINNs to see whether they can be improved. Since the prediction results from $t_0 = 0.1$ to $t_1 = 0.9$ have been attained as shown in Figure 4, we take the approximated solution $\bar{u}(t_1, x_k)$ for $k = 1, \cdots, 2048$ as the real value. Then we feed it into another neural network with the same structure as before in order to predict the solutions at $t_2 = 1.5$. In this trial, all training data are still sub-sampled randomly from the discretised domain. Figure 8 illustrates the results at the final stage. Then, based on the estimated solution at t_1 displayed in Figure 5 using adaptive collocation, 200 training points at t_1 are selected adaptively according to the value of $\bar{u}_a(t_1, x_k)$ for $k = 1, \cdots, 2048$. Prediction of u at the final stage is shown in Figure 9. Though the multi-step discrete time

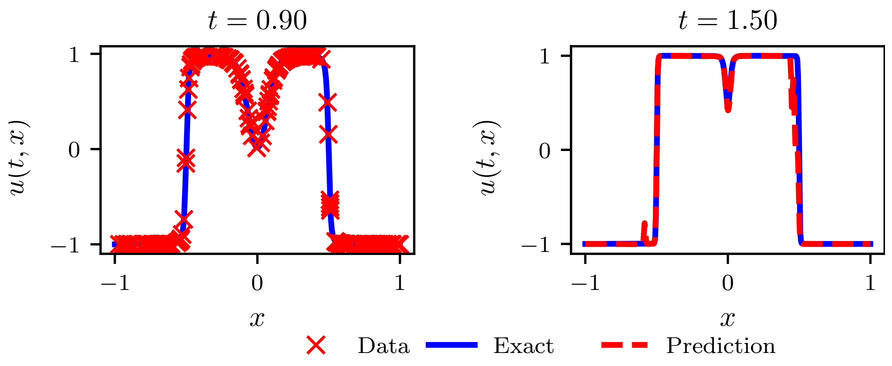

Fig. 8. The left picture depicts the 200 initial training points randomly sub-sampled from the data set of 2048 points at $t_1 = 0.9$, while the right one shows final prediction at $t_2 = 1.5$. The mean relative error on the whole data set is 0.1305.

model still performs badly if sub-sample training data randomly at each step, the model yields desirable outcomes when selecting training points adaptively each time. Hence, adaptive collocation strategy plays a role in improving the accuracy of estimated solutions of the Allen-Cahn equation in longer term.

4 Conclusion

In this study, adaptive collocation strategy is applied in the discrete time models of physics-informed neural networks to estimate solutions of the Allen-Cahn equation. A sample data set of 2048 evenly distributed points in the discretised domain $\Omega = [-1, 1]$ at certain time in discretised time domain $t \in [0, 2]$ is generated using Chebfun package which applies explicit Runge-Kutta integrator. The accuracy of prediction results from $t_0 = 0.1$ to $t_1 = 0.9$ obtained via single-step discrete time model of PINNs shows the capability of the model to find

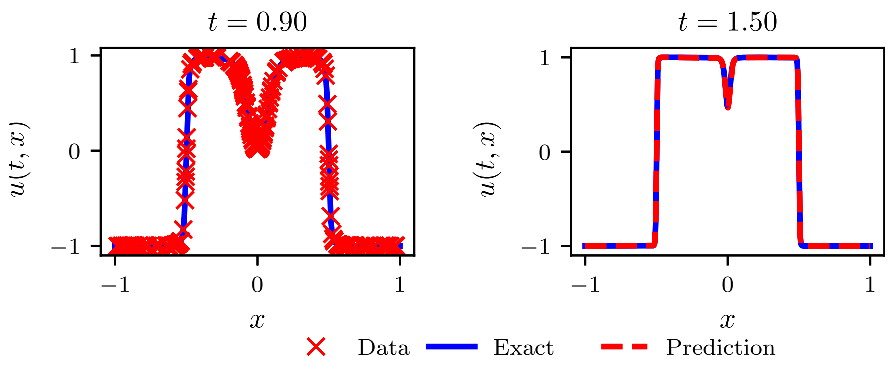

Fig. 9. The left picture depicts the 200 initial training points adaptively selected from the data set of 2048 points based on the value of $\bar{u}(t_1, x_k)$ for $k = 1, \cdots, 2048$, while the right one shows final prediction at $t_2 = 1.5$. The mean relative error on the whole data set is $1.940 \cdot 10^{-3}$.

solutions in short time period. Moreover, the model taking advantage of adaptive collocation approach even generates better simulation outcomes. In long-time prediction, estimation results demonstrate that single-step discrete time models cannot perform well. Through training the model twice with adaptive collocation strategy used to select training points, we provide quite precise approximation results at $t = 1.5$, reflecting the robustness of multi-step discrete time models of PINNs with adaptive collocation in long-time prediction.

When training the models, neural networks with varied hyper-parameters could be employed to see further improvement in the accuracy of prediction, and study of convergence properties of the neural networks remains open as possible research topics. Furthermore, the applicability of multi-step discrete time models of PINNs in other PDEs with larger time domain can be investigated in future work.

References

1. Allen, S.M., Cahn, J.W.: A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. Acta Metallurgica **27**(6), 1085 – 1095 (1979). https://doi.org/https://doi.org/10.1016/0001-6160(79)90196-2, http://www.sciencedirect.com/science/article/pii/0001616079901962
2. Anitescu, C., Atroshchenko, E., Alajlan, N., Rabczuk, T.: Artificial neural network methods for the solution of second order boundary value problems. Computers, Materials and Continua **59**(1), 345–359 (01 2019). https://doi.org/10.32604/cmc.2019.06641
3. Bartels, S., Mller, R., Ortner, C.: Robust a priori and a posteriori error analysis for the approximation of allen-cahn and ginzburg-landau equations past topological changes. Siam Journal on Numerical Analysis **49**(1), 110–134 (2011)
4. Beck, C., Ee, W., Jentzen, A.: Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order

backward stochastic differential equations. Journal of Nonlinear Science **29** (08 2019). https://doi.org/10.1007/s00332-018-9525-3

5. Driscoll, T.A., Hale, N., Trefethen, L.N.: Chebfun Guide. Oxford (2014)
6. E, W., Yu, B.: The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. CoRR **abs/1710.00211** (2017), http://arxiv.org/abs/1710.00211
7. Feng, X., Wu, H.J.: A posteriori error estimates and an adaptive finite element method for the allenccahn equation and the mean curvature flow. Journal of Scientific Computing **24**(2), 121–146 (2005)
8. Gobovic, D., Zaghloul, M.E.: Analog cellular neural network with application to partial differential equations with variable mesh-size. In: Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94. vol. 6, pp. 359–362 vol.6 (1994). https://doi.org/10.1109/ISCAS.1994.409600
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
10. Iserles, A.: A First Course in the Numerical Analysis of Differential Equations. Cambridge Texts in Applied Mathematics, Cambridge University Press, 2 edn. (2008). https://doi.org/10.1017/CBO9780511995569
11. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. CoRR **abs/1711.10561** (2017), http://arxiv.org/abs/1711.10561
12. Shen, J., Yang, X.: Numerical approximations of allen-cahn and cahn-hilliard equations. Discrete and Continuous Dynamical Systems - DISCRETE CONTIN DYN SYST **28**, 1669–1691 (12 2010). https://doi.org/10.3934/dcds.2010.28.1669
13. Sirignano, J., Spiliopoulos, K.: Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics **375**, 1339 – 1364 (2018). https://doi.org/https://doi.org/10.1016/j.jcp.2018.08.029, http://www.sciencedirect.com/science/article/pii/S0021999118305527
14. Suzuki, Y.: Neural network-based discretization of nonlinear differential equations. Neural Computing and Applications **31** (07 2019). https://doi.org/10.1007/s00521-017-3249-4
15. Zang, Y., Bao, G., Ye, X., Zhou, H.: Weak adversarial networks for high-dimensional partial differential equations. Journal of Computational Physics **411**, 109409 (2020). https://doi.org/https://doi.org/10.1016/j.jcp.2020.109409, http://www.sciencedirect.com/science/article/pii/S0021999120301832