Relative Residual Resampling

An Adaptive Residual-Based Sampling Methods in Training PINNs

Master Computer Science

Haoran Wang

Delft University of Technology

TU Delft

Relative Residual Resampling

An Adaptive Residual-Based Sampling Methods in Training PINNs

by

Haoran Wang

Student Name	5225713

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday, July 12, 2024, at 13:00.

Committee:	Prof.dr.ir. M.J.T. Reinders
	Dr. David M. J. Tax
	Dr. Neil Yorke-Smith
	Mahdi Naderibeni
Project Duration:	November, 2023 - July, 2024
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

The thesis contents were enhanced with the assistance of ChatGPT.

Relative Residual Resampling: An Adaptive Residual-Based Sampling Methods in Training PINNs

Haoran Wang[1]

[1]Pattern Recognition Lab, TU Delft

July 5, 2024

Abstract

Physics-Informed Neural Networks (PINNs) offer a promising approach to solving partial differential equations (PDEs). In PINNs, physical laws are incorporated into the loss function, guiding the network to learn a model that adheres to these laws as defined by the PDEs. Training PINNs involves using three types of points: collocation points within the domain of the PDEs, boundary points on the edges of the domain, and initial points at the starting time ($t = 0$). A common challenge in training PINNs is the imbalance between the number of boundaries and initial points compared to collocation points, which can negatively impact the training process. Typically, the number of each type of point is manually set, and the number of collocation points is usually ten times that of boundary and initial points, leading to an uneven loss distribution over time.

Our work introduces a method called Relative Residual Resampling (R3) to address this issue. This method dynamically adjusts the number of each type of training point to ensure a more balanced distribution. Consequently, the loss is more evenly spread across the time domain, enhancing the performance of the PINN. We tested our method on the 1D Heat Equation and 1D Diffusion Equation. The results demonstrate that our approach reduces the overall loss by at least 6%, and balances the loss distribution over time by reducing the loss range for at least 65% compared with the state-of-the-art residual-based resampling strategy. This improvement makes PINNs more accurate and efficient for solving time-dependent PDEs, providing a practical solution for applications where understanding temporal dynamics is crucial.

Keywords: Physics-Informed Neural Network, PDE, Adaptive sampling.

1 Introduction

Physics-Informed Neural Networks (PINNs), introduced by Raissi *et al.* [1], provide a scientific machine learning approach for approximating solutions to partial differential equations (PDEs) [2, 3], has gained more popularity in the machine learning field. PINNs incorporate PDEs that govern the physical laws directly into the loss function of the neural network during training, making it an alternative way of solving PDEs besides numerical methods such as the finite element methods [4, 5].

Over recent years, PINNs have demonstrated empirical success in solving various PDEs across multiple fields,

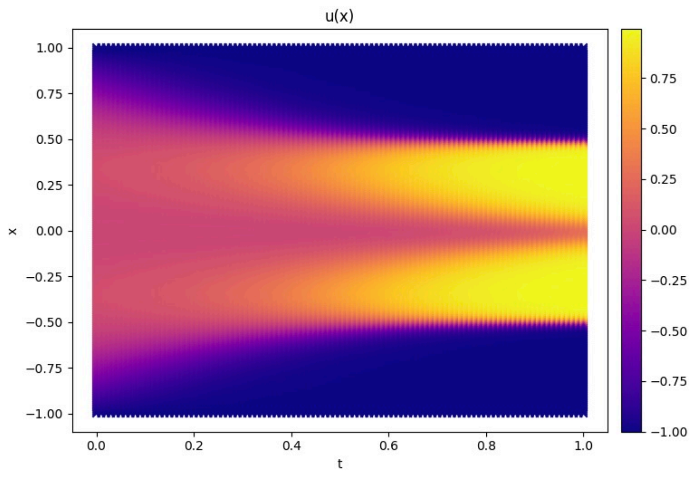


Fig. 1: Solution to 1D Allen-Cahn Equation.

including the Burgers' Equation, convection-diffusion Equation in fluid dynamics [1,6,7], Navier-Stokes equations in fluid mechanics [8], the Wave equation in electromagnetics [9], and the Allen-Cahn Equation used in material science and physics [10]. One example of PINNs in action is solving the 1D Allen-Cahn Equation, a PDE used in material science and physics to model phase separation in multi-component alloy systems. The Allen-Cahn equation describes the temporal evolution of an order parameter, which typically represents the concentration of one component within the alloy, varying over both space and time [11]. Fig. 1 shows the evolution of this solution over time, with yellow areas indicating positive values and purple areas indicating negative values. As time progresses, the difference between these values grows, visually demonstrating the dynamic behavior of the phase separation process.

A key focus in the field of PINNs is the development of effective training algorithms. These algorithms involve selecting sampling strategies, and weighting the loss terms, all of which can significantly impact model accuracy. Training PINNs typically involves three types of points: collocation points within the domain of the PDE, boundary points on the edges, and initial points at the initial time ($t = 0$). PINNs are trained on these three sets of points, to reduce the residual for each point, and learn the model through using the training points. The model's accuracy could be affected by the location of the points, and the number of the points. The loss function is composed of the residual loss (associated with collocation points), initial condition loss, and boundary condition loss. Balancing these components is crucial for the network's performance.

One promising direction in PINN research is re-weighting the loss terms to achieve a lower overall loss. For example, the Neural Tangent Kernel (NTK) approach updates the weight of each loss term based on the eigenvalues of the NTK matrix [9]. Another method involves using a large constant to weight the initial condition loss in time-dependent PDEs [12]. However, these methods often lack flexibility. Wang *et al.* [13] proposed an adaptive scheme that calculates weights using backpropagated gradients, offering more adaptability.

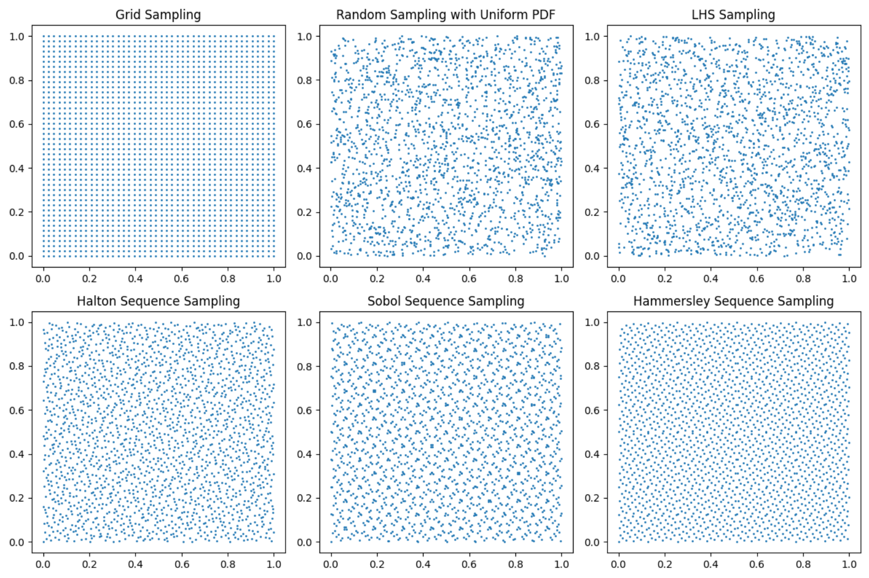


Fig. 2: Uniform Non-Adaptive Sampling Strategies

Sampling strategy also plays a critical role in the training of PINNs. Effective sampling determines where and how many points to sample, significantly impacting the model's accuracy and efficiency. There are two major sampling strategy categories: non-adaptive and adaptive. Non-adaptive sampling strategies fix the distribution of points before training begins, while adaptive sampling strategies adjust the points dynamically during training.

Non-adaptive sampling includes several methods, we include 6 methods in Fig. 2. Uniform grid sampling samples points at regular intervals across the domain, providing an evenly spaced grid of points. Latin Hypercube Sampling (LHS) ensures that each variable is sampled uniformly across its range, which helps in better exploring the input space [14]. The Halton sequence generates quasi-random low-discrepancy points more uniformly distributed than purely random points [15]. Similarly, the Sobol sequence, another low-discrepancy sequence, is often used for high-dimensional integration and sampling due to its uniform distribution properties [16]. The Hammersley sequence also provides quasi-random sampling and is particularly efficient for certain dimensions [17].

On the other hand, adaptive sampling strategies dynamically adjust the sampling points during the training process to improve learning efficiency. For instance, Lu *et al.* [18] proposed Residual-based Adaptive Refinement(RAR): adding points with large residuals to the training set, thereby focusing the learning process on regions where the model's prediction error is high. Wu *et al.* [19] introduced Residual-Based Adaptive Distribution: all training points based on the residual's probability distribution function, further enhancing the network's learning capability by targeting the most challenging areas of the solution domain. Combining both ideas, the authors [19] also proposed to add points based on the probability density function to the original training points set, named residual-based adaptive refinement with distribution (RAR-D).

In this work, we address a significant issue in PINN development: the imbalance of loss distribution across the time domain [20]. Traditional PINN training can disproportionately weight loss terms from different parts of the solution domain, especially in time-dependent problems, hindering network accuracy and convergence. We emphasize the importance of balancing the ratio of training points for initial conditions, boundary conditions, and solving PDEs.

Despite various weighting schemes and adaptive resampling methods, there is a lack of research on determining the optimal number of collocation points and the appropriate ratio of points for initial and boundary conditions. Most studies rely on expert knowledge for setting these numbers, suggesting potential improvements through systematic investigation. To fill this gap, we propose Relative Residual Resampling (R3), a novel approach that dynamically adjusts the number of training points based on the ratio of specific loss terms to the total loss. This method ensures a more evenly distributed loss across the time domain and can be combined with state-of-the-art residual-based adaptive resampling methods (RAD). Our approach enhances PINN performance by maintaining a balanced loss distribution and providing flexibility for integration with other resampling techniques.

2 Background

2.1 Physics-Informed Neural Network

Partial Differential Equations (PDEs) represent the precise physics laws that describe various natural phenomena. Eq. (1) shows its mathematical definition.

$$\mathcal{N}(\boldsymbol{u};\boldsymbol{\theta})(\boldsymbol{x}) = f(\boldsymbol{x}, \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial^2 x_1}, ..., \frac{\partial^2 u}{\partial x_1 \partial x_d}; ...; \boldsymbol{\theta}), \boldsymbol{x} \in \boldsymbol{\Omega} \tag{1}$$

where $\boldsymbol{\Omega} \subseteq \mathbf{R}^d$ is the spatial-temporal domain, $\boldsymbol{x} = (x_1, ..., x_d)$ denotes the coordinates in the spatial-temporal domain $\boldsymbol{\Omega}$, $\boldsymbol{\theta}$ is a collection of all the parameters in the network, and $\boldsymbol{u}(\boldsymbol{x})$ represents the solution to Eq. (1). In time-dependent PDEs, we regard temporal variable t as one component of $\boldsymbol{x}$. Therefore, $\boldsymbol{\Omega}$ also contains the domain defined for time.

The PDEs are also governed by initial conditions (ICs) and boundary conditions (BCs). The initial conditions are defined in Eq. (2), and the boundary condition is defined in Eq. (3).

$$u_{\mathcal{B}}(\boldsymbol{x}) = \mathcal{B}(\boldsymbol{u};\boldsymbol{\theta})(\boldsymbol{x}), \boldsymbol{x} \in \boldsymbol{\partial\Omega} \tag{2}$$

$$u_{\mathcal{I}}(\boldsymbol{x}) = \mathcal{I}(\boldsymbol{u};\boldsymbol{\theta})(\boldsymbol{x}|_{t=0}), \boldsymbol{x} \in \boldsymbol{\partial\Omega} \tag{3}$$

where $\boldsymbol{\partial\Omega}$ is the spatial-temporal domain for ICs and BCs. The PDEs problem is defined as finding $\boldsymbol{u}(\boldsymbol{x})$ given specific BCs and ICs.

The PINN model consists of a neural network, one type of the neural network used in PINN is the feed-forward neural network with hidden layers in between. The feed-forward neural network gets spatial-temporal coordinates $\boldsymbol{x}$ as inputs, and outputs the predicted solution $\boldsymbol{u}(\boldsymbol{x})$ to the PDEs. The feed-forward neural network with MLP architecture is denoted by $\boldsymbol{x}^L : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$. There are L layers in the neural network and each layer l contains of $|\boldsymbol{x}^l|$ neurons, with an input layer denoted in Eq. (4), $L-2$ hidden layer denoted in Eq. (5), and an output layer denoted in Eq. (6).

$$\boldsymbol{x}^0 = \boldsymbol{x}, \qquad \boldsymbol{x} \in \mathbb{R}^{d_{in}} \tag{4}$$

$$\boldsymbol{x}^l = \sigma(\mathbf{W}^l \boldsymbol{x}^{l-1} + \mathbf{b}^l), \qquad l \in [1, L-1], l \in \mathbb{Z},\ \mathbf{W}^l \in \mathbb{R}^{|\boldsymbol{x}^l| \times |\boldsymbol{x}^{l-1}|},\ \mathbf{b}^l \in \mathbb{R}^{|\boldsymbol{x}^l|} \tag{5}$$

$$\boldsymbol{x}^{L-1} = \mathbf{W}^L \boldsymbol{x}^{L-1} + \mathbf{b}^L, \qquad \mathbf{W}^L \in \mathbb{R}^{d_{out} \times |\boldsymbol{x}^{L-1}|},\ \mathbf{b}^L \in \mathbb{R}^{d_{out}}; \tag{6}$$

The loss function of the PINNs is constructed mostly as the mean squared error of the predicted solution on BCs and ICs, and the mean squared error on the collocation points that the solution should satisfy. Eq. (7)-(10) defined the loss term of the loss function in PINNs.

$$\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|} \sum_{\boldsymbol{x} \in \mathcal{T}_f} ||\mathcal{N}(\boldsymbol{x}, u)||^2 \tag{7}$$

$$\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{\boldsymbol{x} \in \mathcal{T}_b} ||\mathcal{B}(u, \boldsymbol{x})||^2 \tag{8}$$

$$\mathcal{L}_i(\boldsymbol{\theta}; \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{\boldsymbol{x} \in \mathcal{T}_i} ||\mathcal{I}(u, \boldsymbol{x})||^2 \tag{9}$$

$$\mathcal{L} = \lambda_f \mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) + \lambda_b \mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b) + \lambda_i \mathcal{L}_i(\boldsymbol{\theta}; \mathcal{T}_i) \tag{10}$$

where $\mathcal{T}_f \in \Omega$ is a set of collocation points, and the residual for the PDEs are defined in Eq. (7). Given a set of collocation points: $\mathcal{T}_b \in \partial\Omega$, the residual for the BCs are defined in Eq. (8), and similarly, the residual for the ICs are defined in Eq. (9). The loss function for PINNs is defined as the sum of these three losses with the pre-defined or adaptive learned parameters λ in Eq. (10). In vanilla PINN [1], the authors take $\lambda_f = \lambda_b = \lambda_i = 1$. PINNs are designed to solve partial differential equations (PDEs) given different BCs and ICs. The problem definition is shown in Eq. (1) - (3). This is a generic setup for PDEs to solve using PINNs. PINNs consist of three parts: a feed-forward neural network, a loss function embedded with physics information using PDE residuals, BCs, and ICs, and an activation step with a minimization step of the loss concerning the weights and biases in the neural network. We define PINNs framework in the following steps:

In the first step, a feed-forward neural network is being constructed with parameters $\boldsymbol{\theta} = \{\mathbf{W}^l, \mathbf{b}^l\}_{1 \le l \le L}$. The inputs are the training data $\boldsymbol{x}$ representing the spatiotemporal coordinates in training sets $\mathcal{T}_f$, $\mathcal{T}_b$, and $\mathcal{T}_i$, and the output is of the same dimension as $\boldsymbol{u}$ being the solution to the PDEs. The neural network is suitable for solving PDEs because the natural representation of partial derivatives in the neural network could be calculated using automatic differentiation.

The second step of the algorithm requires three training sets to be specified for the training process. In [18], the authors regard the BCs and the ICs as the same training set since ICs could be categorized as special

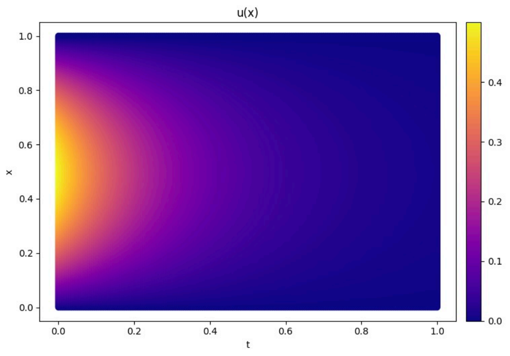


Fig. 3: 1D Heat Equation defined on $(x, t) \in [0, 1] \times [0, 1]$

boundary conditions given time $t = 0$.

The third step is defining and calculating the loss function. Here we use a generic definition of the loss function in Eq. (7) to demonstrate the workflow of the algorithm and notice that it also varies and deviates from the loss function we defined in the algorithm. The loss function embeds the physical information inside PINNs because it shows the direction of optimizing $\boldsymbol{\theta}$ when minimizing loss.

The last step is to optimize $\boldsymbol{\theta}$ to minimize the loss. This is the training step of PINNs, and different PINN framework has different training methods. PINNs are generally being trained by minimizing the loss function with gradient-based optimizers, such as gradient descent, Adam optimizer [21], and L-BFGS optimizer [22]. The Adam optimizer and L-BFGS optimizer have their strengths and weaknesses regarding different types of PDEs. For smooth PDE solutions, L-BFGS performs better due to it utilizes the second-order derivatives while the Adam optimizer only uses the first-order derivatives. However, in stiff PDE solutions, the Adam optimizer performs better since L-BFGS could be stuck at a local minimum.

2.2 Example PDE problems

We choose three example PDE problems: 1D Heat Equation, 1D Diffusion Equation, and 1D Burgers' equation. These equations are used to test the PINNs later.

2.2.1 Heat Equation

The 1D Heat Equation describes the temperature at a point x along the rod at time t. We set Ω to be the interval $[0, 1] \times [0, 1]$ in the testing 1D Heat Equation. The parameter α typically represents the thermal

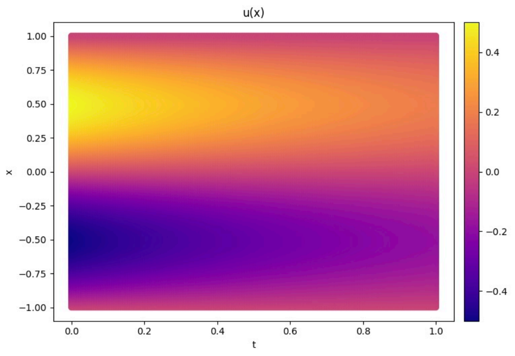


Fig. 4: 1D Diffusion Equation defined on $(x,t) \in [-1,1] \times [0,1]$

diffusivity, which governs the heat transfer rate and diffusion through a material.

$$\begin{aligned} \frac{\partial u}{\partial t} &= \alpha \frac{\partial^2 u}{\partial x^2}, \quad x \in [0,1], \quad t \in [0,1], \\ u(x,0) &= -\sin(\pi x), \\ u(0,t) &= u(1,t) = 0. \end{aligned} \tag{11}$$

The term $\frac{\partial u}{\partial t}$ represents the time derivative, meaning the rate of change of temperature concerning time at a specific point x. The second-order term $\frac{\partial^2 u}{\partial x^2}$ describes how the temperature is changing concerning space. Specifically, it represents the curvature or the concavity of the temperature profile along the rod. High curvature indicates regions where heat is likely to flow more rapidly. The constant α (thermal diffusivity) quantifies how quickly heat diffuses through the material of the rod. It combines the material's conductivity, density, and specific heat capacity into a single parameter.The exact solution to Eq. (11) is:

$$u(x,t) = e^{-\pi^2 \alpha t} sin(\pi x) \tag{12}$$

2.2.2 Diffusion Equation

The 1D Diffusion Equation describes how a quantity $u(x,t)$ (which could represent temperature, concentration of a substance, etc.) evolves in space due to diffusion and an additional source term. We set Ω to be the interval $[-1,1] \times [0,1]$ in the testing 1D Diffusion Equation.

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2} - e^{-t}(sin(\pi x) - \pi^2 sin(\pi x)), \quad x \in [-1,1], \quad t \in [0,1], \\ u(x,0) &= \sin(\pi x), \\ u(-1,t) &= u(1,t) = 0. \end{aligned} \tag{13}$$

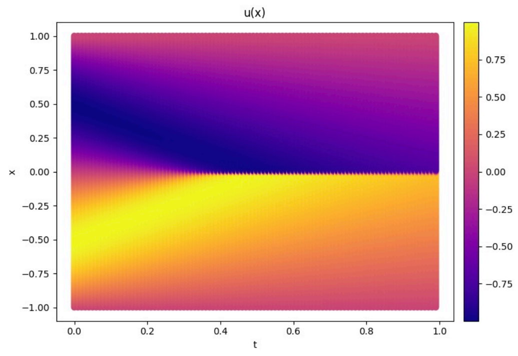


Fig. 5: 1D Burgers' Equation defined on $(x,t) \in [-1,1] \times [0,1]$

This term $\frac{\partial u}{\partial t}$ represents the rate of change of u for time. It indicates how u is evolving at each point x as time progresses. This term $\frac{\partial^2 u}{\partial x^2}$ describes the spatial diffusion of u. It represents how the quantity u spreads out over space, driven by the gradient of u.This term $e^{-t}(sin(\pi x) - \pi^2 sin(\pi x))$ adds or removes the quantity u at different points in space and time. The source term is a function of both space x and time t (the source term), indicating a time-dependent addition to the diffusion process.The exact solution to Eq. (13) is:

$$u(x,t) = e^{-t} sin(\pi x) \tag{14}$$

2.2.3 Burgers' Equation

The 1D Burgers' equation models the evolution of a velocity field $u(x,t)$ in a fluid-like system, combining elements of both diffusion and convection. We set Ω to be the interval $[-1,1] \times [0,1]$ in the testing 1D Burgers' Equation. The parameter ν represents the kinematic viscosity coefficient: a measure of the fluid's resistance to flow under the influence of gravity.

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} &= \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1,1], \quad t \in [0,1], \\ u(x,0) &= -\sin(\pi x), \\ u(-1,t) &= u(1,t) = 0. \end{aligned} \tag{15}$$

This term $\frac{\partial u}{\partial t}$ represents the rate of change of the velocity for a time at a specific point x. It indicates how the velocity field is evolving. This nonlinear term $u\frac{\partial u}{\partial x}$ represents the self-advection of the velocity field. It describes how the velocity at a point influences the rate of change of velocity at that point. This term is responsible for the transport of the velocity field by itself, leading to nonlinear effects such as shock formation and steepening of waves. The term $\nu\frac{\partial^2 u}{\partial x^2}$ represents the viscous diffusion of the velocity field, where ν is the kinematic viscosity. This term tends to smooth out the velocity field over time, counteracting the steepening effects of the convection term.

3 Methodology

To balance the loss across the time domain and to enhance the model accuracy, we propose Relative Residual Resampling (R3) in Algorithm 1. The algorithm started with selecting a starting round of points $\mathcal{T} = \mathcal{T}_r \cup \mathcal{T}_b \cup \mathcal{T}_i$, setting each subset to the same size $\frac{N}{3}$, and training the model for k iterations. In the second step, we use the Monte Carlo Integration method by randomly sampling three sets of points, and calculating different evaluation functions to estimate residual, initial condition error, and boundary condition error. Note that for residuals, we use the current model learned by PINN, which serves as an estimation of the residual within the domain. For IC error and BC error, we use the error between the estimated value and the exact BC/IC value. We also set a constant number $C = 100$ to be the smallest number of points possible in each set in Eq. 19-21. After estimating the error for residual, IC, and BC, we will calculate the new number of points based on their residuals. Notice the number of points remains fixed at N. The second to the last step is to randomly resample the points with a uniform probability density function based on the new number calculated in the previous step. Then we train the network for another k iterations until the total number of iterations reaches the limit.

The residual error represents the error associated with the training points within the domain, excluding the points on the initial and the boundary conditions. To give a better understanding of this error, we refer to the Eq. (16). In the nth round, we estimate the error by feeding the testing points to the function learned by the model in the previous $n-1$ rounds and calculate the mean L2-Norm value of the residual. This error could be calculated more accurately, given the true value of the testing points and the predicted value of the training points. However, this is not the case if the solution to the PDE is unknown. To estimate the boundary condition error and the initial condition error, we could use the exact boundary and initial condition, since it is given when the PDE problem was defined. In Algorithm 1, we calculate the boundary condition error by Eq. (17). and calculate the initial condition error by Eq. (18). The difference between the calculation of the boundary condition error, and the initial condition error with the residual error is that we have an exact boundary and initial condition, making it possible for us to calculate the exact error for each training point.

The error calculated by Eq. (16-18) is the approximation error between the solution $\hat{u}_{\mathcal{T}}$ generated by the neural network and the best solution could generated by the neural network $u_{\mathcal{T}}$. $u_{\mathcal{T}}$ could also be interpreted as the solution generated by the neural network with the global minimum loss. To illustrate this, consider three errors mentioned in Fig. 6 proposed by Lu *et al.* [18]: the generalization error between the PDE solution u to the best solution u_F generated in the domain F, the optimization error between u_F and the best solution that could be generated by the neural network $u_{\mathcal{T}}$, and the approximation error between $u_{\mathcal{T}}$ and the solution $\hat{u}_{\mathcal{T}}$ generated by the neural network.

In R3, or other sampling methods, the choice of calculating the resampling indicator (in our case is ϵ_r, ϵ_b, and ϵ_i defined in Eq. (19-21) could be different. In the Eq. (16-18), we used *mean* L^2 *norm* to calculate the resampling indicator. Another variation of R3 is that the algorithm stays the same as Algorithm 1, while the resampling indicators are changed to L^2 *norm*. The L^2 *norm* of $\boldsymbol{x} = \{x_i\}_{i=1}^n$is defined as:

$$||x||_2 = \sqrt{\sum_i^n x_i^2} \tag{22}$$

Algorithm 1 Relative Residual Resampling (R3)

```
1: **Input:** Hyperparameters: $N = 6000, C = 100, K = 20000, k = 1000, m = 100000$
2: **Output:** Trained PINN model
3: Sample the initial training points $\mathcal{T} = \mathcal{T}_r \cup \mathcal{T}_b \cup \mathcal{T}_i$, The number of points in each set is set as the same
   number: $|\mathcal{T}_r| = |\mathcal{T}_b| = |\mathcal{T}_i| = \frac{N}{3}$
4: Train the PINN for $k$ iterations
5: **repeat**
6:     $\mathcal{S} \leftarrow m$ collocation points randomly sampled within the domain according to the uniform PDF
7:     $\mathcal{S}_b \leftarrow m$ points randomly sampled on boundary according to the uniform PDF
8:     $\mathcal{S}_i \leftarrow m$ points randomly sampled on domain with $t = 0$ according to the uniform PDF
9:     Estimate the residual by Monte Carlo integration:
           $$\epsilon_r = \frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{x} \in \mathcal{S}} \left\| f\left(\boldsymbol{x}; \frac{\partial \hat{u}}{\partial x_1}, ..., \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, ..., \frac{\partial \hat{u}}{\partial x_1 \partial x_d}; ...; \lambda\right) \right\| \quad (16)$$
10:    Estimate the BC error:
           $$\epsilon_b = \frac{1}{|\mathcal{S}_b|} \sum_{\boldsymbol{x} \in \mathcal{S}_b} \| \hat{u}(\boldsymbol{x}, \theta) - \hat{u}_{\mathcal{B}}(\boldsymbol{x}) \| \quad (17)$$
11:    Estimate the IC error:
           $$\epsilon_i = \frac{1}{|\mathcal{S}_i|} \sum_{\boldsymbol{x} \in \mathcal{S}_i} \| \hat{u}(\boldsymbol{x}|_{t=0}, \theta) - \hat{u}_{\mathcal{I}}(\boldsymbol{x}|_{t=0}) \| \quad (18)$$
12:    Calculate the new number of collocation points to be sampled:
           $$N_r = (N - 3C) \frac{\epsilon_r}{\epsilon_r + \epsilon_b + \epsilon_i} + C \quad (19)$$
13:    Calculate the new number of BC points to be sampled:
           $$N_b = (N - 3C) \frac{\epsilon_b}{\epsilon_r + \epsilon_b + \epsilon_i} + C \quad (20)$$
14:    Calculate the new number of IC points to be sampled:
           $$N_i = (N - 3C) \frac{\epsilon_i}{\epsilon_r + \epsilon_b + \epsilon_i} + C \quad (21)$$
15:    $\mathcal{T}_r \leftarrow N_r$ collocation points randomly sampled within the domain according to the uniform PDF
16:    $\mathcal{T}_b \leftarrow N_b$ points randomly sampled on boundary according to the uniform PDF
17:    $\mathcal{T}_i \leftarrow N_i$ collocation points randomly sampled within the domain according to the uniform PDF
18:    Train the PINN for $k$ iterations
19: **until** the total number of iterations reaches the limit $K$
20: **Return:** Trained PINN model
```

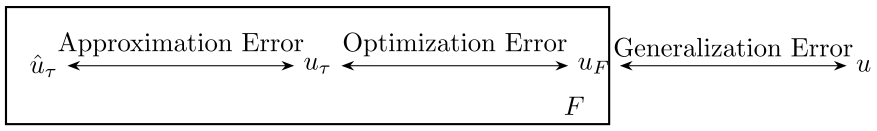


Fig. 6: Illustration of errors of a Physics-Informed Neural Network (PINN).

The mean L^2 *norm* for a discrete set of values $\{x_i\}_{i=1}^n$ is defined as:

$$\|x\|_{2mean} = \sqrt{\frac{1}{n}\sum_i^n x_i^2} \tag{23}$$

L^2 *norm* and the *mean* L^2 *norm* are different by a factor of $\sqrt{\frac{1}{n}}$. In Equation 16-18, to use L^2 *norm* or the *mean* L^2 *norm* are the same if given $n = \frac{1}{|\mathcal{S}|} = \frac{1}{|\mathcal{S}_b|} = \frac{1}{|\mathcal{S}_i|}$ since both the numerator and the denominator times a factor of $\sqrt{\frac{1}{n}}$ will transform the Equation 16-18 from using *mean* L^2 *norm* to L^2 *norm*.

It is also possible to combine R3 with other sampling methods, such as residual-based adaptive distribution (RAD) introduced in [19]. R3 is a resampling scheme based on the estimated residual during training by using Monte Carlo integration, meaning that only the manipulation of the number of points sampled within the domain, on the boundary and the initial condition, but not the specific location of the training points. Therefore, combining methods specific to the location of the training points such as RAR [18], or RAD introduced in [19] is possible. Eq. (24) shows an example of combining the R3 method with RAD by simply changing the step 15-17 to Eq. (24). We resample collocation points according to the PDF of Eq. (24). The error function on the numerator of the Eq. (24) is as same as the error function we used in the Eq. (16). The constant c is a hyper-parameter suggests by Lu *et al.* [18].

One fundamental question to ask is whether R3 differs from loss re-weighting since both use estimated error as the resampling indicator. There is a difference between loss re-weighting and changing the number of points in R3, since the loss term is changing while the number of points is not changing in the first set of methods, the number of points changing in R3 increases the chance of lowering the loss in a high-residual area if given the same PDF for the training points sampled in both loss re-weighting and the R3. We define adaptive weight resampling for comparison with R3 in [23].

$$p(\boldsymbol{x}) \propto \frac{\|f(\boldsymbol{x}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_1 \partial x_d}; \ldots; \lambda)\|}{\epsilon_r} + c \tag{24}$$

4 Results

4.1 Experiments Setup

Table 1: Experimental Setup for Comparing Different Resampling Methods

Network Settings	
Network size	3×20
Optimizer	Adam
Activation Function	tanh
Learning rate	0.001
Initial Number of Points	
Collocation Points	2000
Boundary Condition Points	2000
Initial Condition Points	2000
Constant C in Eq. (14) - (16)	100
Training Duration	20000 Epoch
Comparision Metrics	L^2 Relative Error

We compare Relative Residual Resampling (R3) and Relative Residual Resampling (R3) with RAD to the state-of-the-art re-sampling method RAD, together with RAR, RAR-D (Residual-based adaptive refinement with distribution), random resampling with uniform PDF (Random-Uniform-R), and adaptive weight re-sampling. These 7 methods are compared to solving 3 forward PDE problems: the 1D Heat Equation, 1D Diffusion Equation, and 1D Burgers' Equation. We summarize the experiment settings in the Table 1.

For the R3, the number of points changes dynamically according to the Algorithm 1, and for the R3 with RAD, the number of points changes based on Algorithm 2. For other methods, the number of points stays the same, except for the RAR, and RAR-D, because they are adding 10 points to the large residual area every $k = 1000$ epochs. We first train the network for 1000 epochs instead of what authors did in [19] for $15,000$ epochs. Then we resample the points every 1000 epochs. For each resampling method, either point is being added every 1000 epoch such as RAR, RAR-D, or points are being resampled such as Random-Uniform-R, or RAD. Note that R3 with RAD is a combination of both. We compare the results after 10 rounds of experiments running the experiments for $20,000$ epochs. All the results have an average value of 10 rounds.
To evaluate the accuracy of the prediction $\hat{u}_\tau$ by using PINNs, we adopt the L^2 relative error:

$$\frac{||\hat{u}_\tau - u||_2}{||u||_2} \tag{25}$$

4.2 Loss comparison

As shown in Table 1, R3 achieved better accuracy for solving the Burgers', Heat, and Diffusion Equation compared to the state-of-the-art methods RAD and its variation RAR-D. R3 with RAD outperformed RAD in solving Burgers', Heat, and Diffusion Equation. R3 achieved the best result for solving Burgers' and Diffusion Equations.

Sampling Methods	Burgers	Heat	Diffusion
Random-Uniform-R	$1.54 \times 10^{-1} \pm 4 \times 10^{-3}$	$1.040 \times 10^{-1} \pm 4.0 \times 10^{-3}$	$3.04 \times 10^{-2} \pm 3.0 \times 10^{-4}$
Adaptive Weight Resampling	$2.22 \times 10^{-1} \pm 1.5 \times 10^{-2}$	$1.510 \times 10^{-1} \pm 2.0 \times 10^{-3}$	$2.74 \times 10^{-2} \pm 2.0 \times 10^{-4}$
RAR	$2.10 \times 10^{-1} \pm 4 \times 10^{-3}$	$\mathbf{4.85 \times 10^{-2} \pm 1 \times 10^{-4}}$	$2.62 \times 10^{-2} \pm 4.0 \times 10^{-4}$
RAD	$1.70 \times 10^{-1} \pm 1.0 \times 10^{-2}$	$6.13 \times 10^{-2} \pm 1 \times 10^{-4}$	$5.11 \times 10^{-2} \pm 2.00 \times 10^{-3}$
RAR-D	$\underline{1.47 \times 10^{-1} \pm 7 \times 10^{-3}}$	$8.67 \times 10^{-2} \pm 3.0 \times 10^{-3}$	$\underline{2.09 \times 10^{-2} \pm 1.0 \times 10^{-4}}$
Relative Residual Resampling (R3)	$\mathbf{1.39 \times 10^{-1} \pm 7 \times 10^{-3}}$	$\underline{5.24 \times 10^{-2} \pm 4 \times 10^{-4}}$	$\mathbf{1.97 \times 10^{-2} \pm 0.0 \times 10^{-4}}$
R3 with RAD	$\underline{1.49 \times 10^{-1} \pm 1 \times 10^{-3}}$	$\underline{5.07 \times 10^{-2} \pm 1 \times 10^{-4}}$	$\underline{2.33 \times 10^{-2} \pm 2.0 \times 10^{-4}}$

Table 2: Loss values (mean ± variance) for different PDEs and rounds

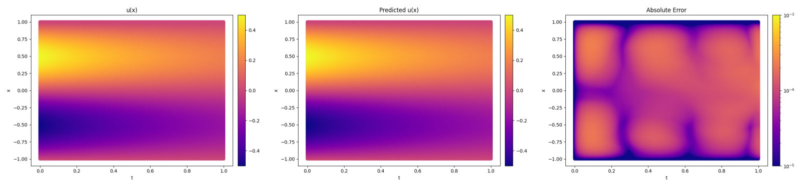
It is noted that RAD is not necessarily the best method when compared with RAR, RAR-D, or even Random-Uniform-R for solving these 3 PDEs. RAR outperformed other methods in solving heat equations. However, we notice that R3 and R3 with RAD are the second and third best methods, and with only a 4% difference between RAR with R3 with RAD.

The loss between each method is relatively small (less than 10%) for the first two best-performing methods. For instance, 5.7% and 6.1% for the difference between RAR-D with R3 solving the Burges' and the Diffusion equation.

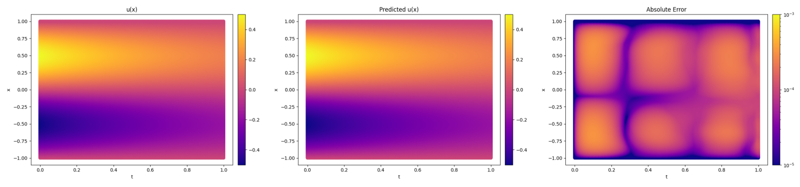
The conclusion from this loss comparison would be RAD is not the best sampling method under this experiment setting, and R3 could achieve a similar performance to the state-of-the-art accuracy in solving Burgers', Heat, and Diffusion Equation under the given experiment setting. The difference between the best two performing methods is less than 10%, showing the average accuracy does not differ too much between the best two performing methods.

The loss in Table 1 represents the mean value after running the experiments for 10 rounds. We also show results in the entire domain in Fig. 7 for solving diffusion equation and Fig. 8 for solving heat equation. In Fig. 7, we observe that R3 and R3 with RAD learned the PDE solution with less error at $t \approx 0.2, 0.6$ and 0.9. At the same time, Random-Uniform Resampling, and RAD failed to learn the solution with more details. In Fig. 8, R3 and R3 with RAD perform better than the other two methods. This is because a lower loss value is learned within the domain, and on the boundary as well as at the initial time. A more detailed pattern is learned within the domain for both PDEs by using R3 and R3 with RAD. The lower the loss represents the more information about the PDE solution is being learned. We can learn that R3 or R3 learned a lower loss by learning well on both boundaries, initial time, and inside the domain. In Fig. 8(b), after 18000 epoch, R3 with RAD learned a similar result compared with R3 in Fig. 8(a), but with only 8000 epoch. This convergence analysis will be mentioned in next subsection.

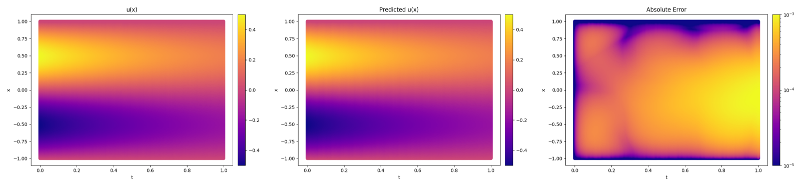
However, we note that obtaining a low L^2 relative error is not necessarily the indicator of a better sampling strategy, and a detailed loss across time-domain analysis will be made in the next subsection.



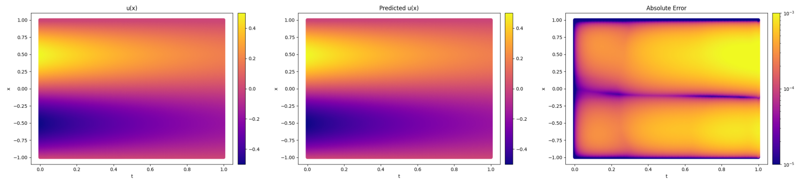
(a) R3



(b) R3 With RAD

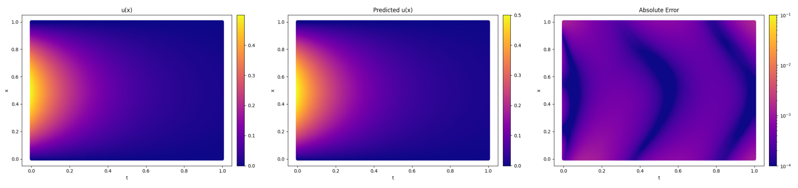


(c) Random-Uniform Resampling

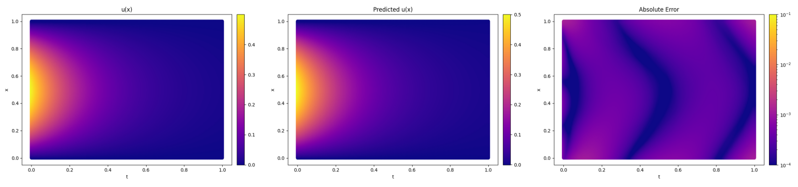


(d) RAD

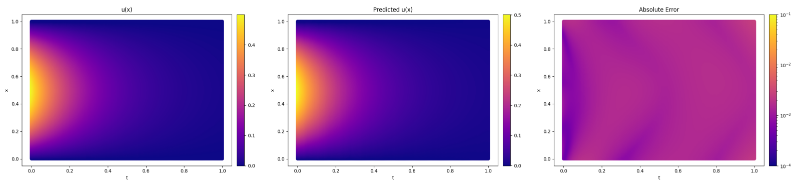
Fig. 7: One-Dimensional Diffusion Equation: (a) The predicted solution versus the exact solution using R3 after 18000 Epoch. The relative error is $0.0.197 \pm 0.000004$. (b) The predicted solution versus the exact solution using R3 with RAD after 8000 Epoch. The relative error is 0.0233 ± 0.00002. (c) The predicted solution versus the exact solution using Random-Uniform Resampling after 18000 Epoch. The relative error is 0.0304 ± 0.0003. (d) The predicted solution versus the exact solution using RAD. The relative error is 0.0511 ± 0.002 after 18000 Epoch.



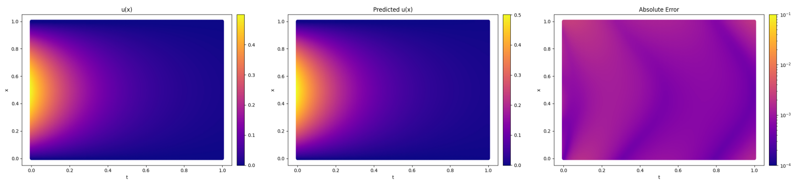
(a) R3



(b) R3 With RAD



(c) Random-Uniform Resampling



(d) RAD

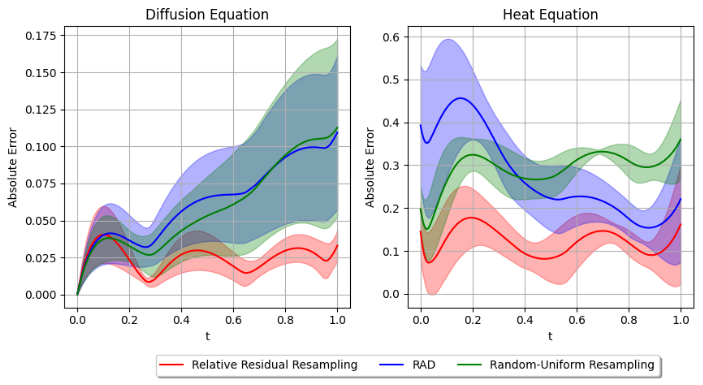
Fig. 8: One-Dimensional Heat Equation: (a) The predicted solution versus the exact solution using R3 after 16000 Epoch. The relative error is 0.0524 ± 0.0004. (b) The predicted solution versus the exact solution using R3 with RAD after 16000 Epoch. The relative error is 0.0507 ± 0.0001. (c) The predicted solution versus the exact solution using Random-Uniform Resampling after 16000 Epoch. The relative error is 0.104 ± 0.004. (d) The predicted solution versus the exact solution using RAD after 16000 Epoch. The relative error is 0.0613 ± 0.0001.

4.3 Loss across time-domain analysis

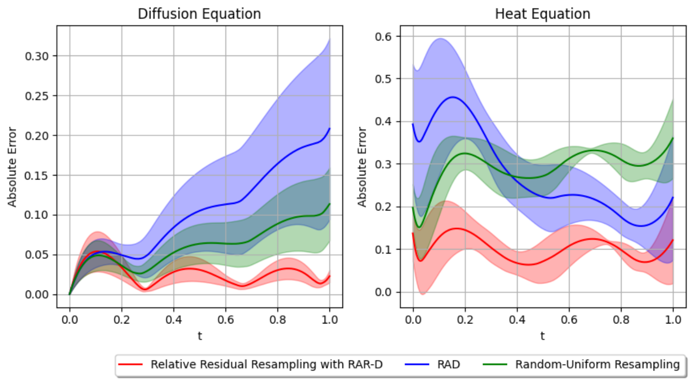
The uneven distribution of loss across the time domain for solving PDEs is one of the major issues in the training process of PDEs. The evenly distributed loss suggests PINNs learned heavier on some part of the time domain, while others remain less learned, or even unlearned. In this section, we will show that after only less than 20000 iterations, R3 started to lower the loss either at the beginning or at the end of the time domain. The loss we are using in this section is the absolute loss defined as:

$$|\hat{u}_\tau - u| \tag{26}$$

4.3.1 Temporal Balanced Loss Distribution



(a) R3, Random-Uniform-R and RAD



(b) R3 with RAD, Random-Uniform-R and RAD

Fig. 9: (a) Comparing the loss on the time domain for the predicted solution using R3 with that of Random-Uniform Resampling and RAD solving Diffusion Equation at 18000 epoch, and Heat Equation at 16000 epoch. (b) Comparing the loss on the time domain for the predicted solution using R3 with RAD with that of Random-Uniform Resampling and RAD solving Diffusion Equation at 8000 epoch, and Heat Equation at 16000 epoch. The shading area represents the mean ± standard deviation.

A key advantage of R3 is its ability to achieve a more balanced loss distribution across the time domain.

In Fig. 9(a), we compare R3, Random-Uniform-R, and RAD in solving diffusion equations and heat equations, and in Fig. 9(b) we compare R3 with RAD, Random-Uniform-R, and RAD in solving the same two equations. Regarding both equations, both R3 and R3 with RAD achieved the smallest loss, and it is more balanced across the entire time domain because the range between the maximum loss and the minimum loss for R3 and R3 with RAD is the smallest among the three methods. We also noticed that RAD achieved the worst performance because it is the most unbalanced method with the largest range among the three. The loss for RAD is either small at $t \approx 0$ or $t \approx 0.9$, and the loss is large on the other end. In Fig. 9(a), RAD solved diffusion equation with a large error at $t \approx 1$, and solved heat equation with a large error at $t \approx 0$. Similar behavior could be observed in Fig. 9(b).

This suggests RAD either learned a smaller loss at the initial time, or at the end of the time, but not both. However, R3 achieved a more balanced loss with both ends and in between. What is more, because of a smaller loss, we observe the solution could be learned well from a local perspective, meaning that the solution to PDEs could be learned in more detail because the peaks are more obvious when we observe the loss curve. To explain this, let's focus on Fig. 7(a) and Fig. 8(b), because a smaller loss is learned in both plots, the more local pattern of the solution could be shown in the plot, giving a clearer pattern of using R3 or R3 with RAD. The results are also consistent comparing both Fig. 9(a) with Fig. 7, and comparing Fig. 9(b) with Fig. 8 since we used the data from the consistent epoch number for comparison in three plots.

4.3.2 Convergence Behavior

R3 and R3 with RAD show distinct convergence behaviors contributing to their effectiveness in solving PDEs. In Fig. 10, the purple and blue curves represent training and testing loos of using R3 and R3 with RAD, respectively. It converges faster than any other method after the 2500 epoch for the training loss. The convergence behavior of R3 is also evident from the decreasing loss range over time. Initially, at 2000 epochs, the loss range is wide, but it narrows by 16000 epochs, indicating steady convergence, as shown in Fig. 11. R3 with RAD demonstrated a faster convergence speed for the diffusion equation by reducing the residual value at peaks and achieving faster loss value convergence, as seen in Fig. 12. It is worth noticing that for the first 8000 epochs of using R3 with RAD, it reduced to approximately the same level as R3. The convergence speed for both R3 and R3 with RAD is faster than the other two methods. We also remark that in Fig. 12, R3 and R3 with RAD started with an even higher loss but gradually decreased to the smallest among the three methods. R3 with RAD even shows its decrease of loss behavior as early as in the 6000 epoch. Similar behavior is also observed in Fig. 11. We can see that R3 already outperformed the other two methods at the 8000 epoch, and it continues to decrease the loss in the 16000 epoch, while the other two methods remain the similar value compared to 8000 epoch with 16000 epoch in Fig. 11(a).

R3 and R3 with RAD demonstrate their speed of convergence is faster than RAD and Random-Unfirom-R, while also showing a continued converge trending in the first 18000 epoch, or even earlier in solving both Diffusion Equation and Heat Equation. This gives us the benefit of training network with shorter iterations while achieving a similar performance that a network would achieve without using R3.

4.3.3 Efficient Point Utilization

R3 and R3 with RAD demonstrate efficient utilization of residual and IC/BC points, contributing to their improved performance. In the heat equation analysis, R3 and R3 with RAD used significantly more residual points than IC/BC points, with the number of IC/BC points converging after 4000 epochs, as observed

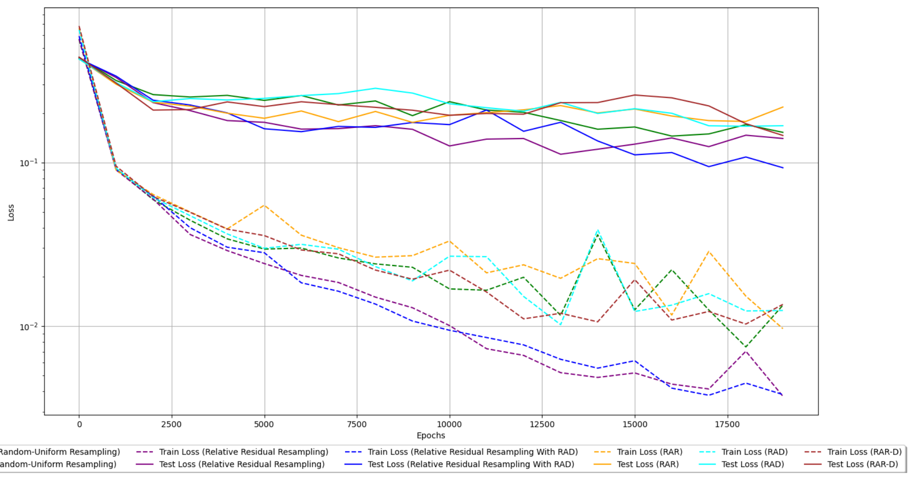
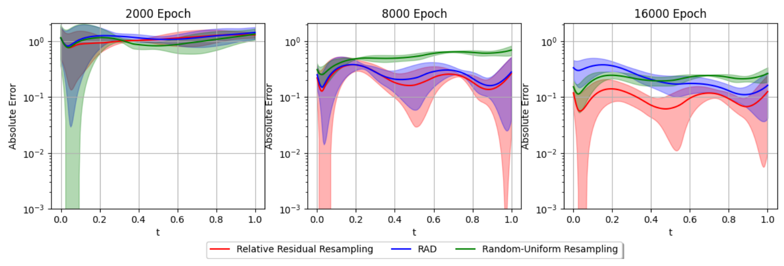
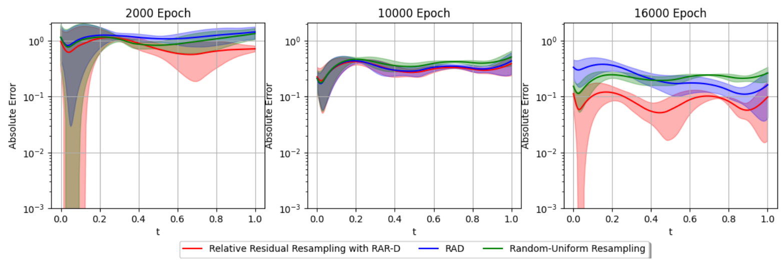


Fig. 10: Training and Testing Curve solving Burgers' Equation.

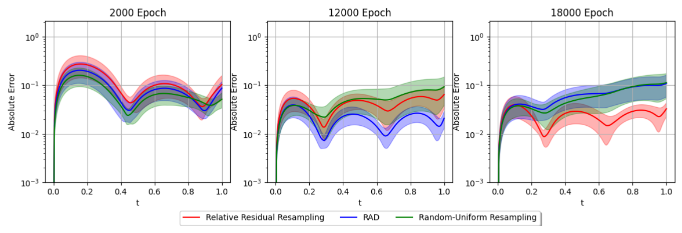


(a) R3, Random-Uniform-R and RAD

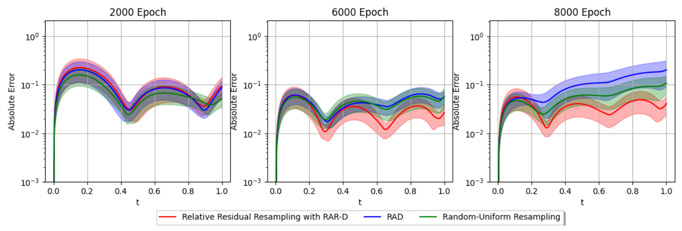


(b) R3 with RAD, Random-Uniform-R and RAD

Fig. 11: One-Dimensional Heat Equation: (a) Comparing the loss on the time domain for the predicted solution using R3 with that of Random-Uniform Resampling and RAD. (b) Comparing the loss on the time domain for the predicted solution using R3 with RAD with that of Random-Uniform Resampling and RAD. The shading area represents the mean ± standard deviation.



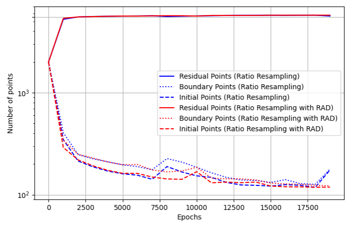
(a) R3, Random-Uniform-R and RAD



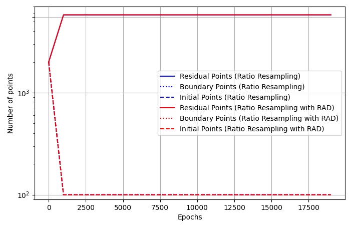
(b) R3 with RAD, Random-Uniform-R and RAD

Fig. 12: One-Dimensional Diffusion Equation: a) Comparing the loss on the time domain for the predicted solution using R3 with that of Random-Uniform Resampling and RAD. (b) Comparing the loss on the time domain for the predicted solution using R3 with RAD with that of Random-Uniform Resampling and RAD. The shading area represents the mean ± standard deviation.

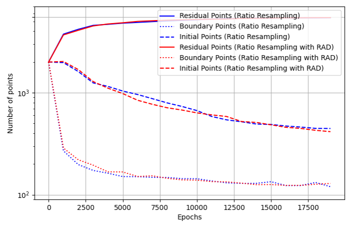
in Fig. 13(a). This efficient allocation of points was also seen in the diffusion equation, where the number of points remained static after 1000 epochs, indicating a balanced and optimized number of points for residual and IC/BC, as shown in Fig. 13(b). In Burgers' equation, R3 showed a decreasing yet large number of IC points (around 600 points) at 8000 epochs, while the number of boundary points converged to around 150 points, suggesting a well-learned boundary, as depicted in Fig. 13(c).



(a) 1D Heat Equation



(b) 1D Diffusion Equation



(c) 1D Burgers' Equation

Fig. 13: Number of collocation, boundary, and initial points using R3 and R3 with RAD.

What is worth mentioning is the expert experience in setting the number of points. We observe a trend of using more and more collocation points and fewer IC/BC points in all three subplots in Fig. 13. We also see that the diffusion equation has only 100 IC and 100 BC points after 1000 epochs. This is the setting that is close to the expert's setting for training PINNs (which is to use only a few hundred points for IC/BC), but not quite the same because we started with 2000 points each. One possible explanation that R3 performs better in balancing the loss across time could be the knowledge of the boundary and IC learned in the early epoch before IC//BC points converged to less than 200 points. The expert's setting is not necessarily the best setting, even given the fact that the convergence of the number of points is close to the expert's setting. We could also argue that R3 performs better in solving Burgers' equation than RAD but with more than 400 IC points even after 17000 epoch. This setting is not necessarily the best setting, but it is better than the expert setting because we demonstrate that R3 achieved a more balanced loss and a lower loss than RAD and the baseline Random-Uniform-Resampling.

To conclude the result section, Relative Residual Resampling (R3) demonstrates significant improvements in balancing loss distribution and improving accuracy across different PDEs compared to traditional methods such as Random-Uniform Resampling and RAD. R3 with RAD consistently achieves lower losses and better accuracy, particularly in the initial stages of training. The efficient utilization of points and distinct convergence behavior further enhance R3's effectiveness. Despite challenges in learning initial conditions, R3 shows promise as a robust method for enhancing the training of PINNs.

5 Discussion

We demonstrate the promising result for using R3 as a novel sampling technique to balance the loss between the residual within the domain, the IC, and the BC. R3 achieved a lower loss in solving Burgers' Equation and Diffusion Equation and also achieved the second smallest loss compared to the best result among 7 different sampling techniques in solving the Heat Equation. We notice that R3 could achieve state-of-the-art accuracy while making improvements on balancing the loss across the time domain in solving Heat and Diffusion equations. However, there are a few topics that are not being studied within this report, and they should be addressed in the future to better understand the relationship between the number of sampling points and the value of the loss for collocation points, IC, and BC.

R3 uses less than 2000 points in solving the Burgers' Equation, while the baseline Random-Uniform Resampling used 2000 points and achieved a more balanced loss and a smaller IC loss at $t = 0$ in Fig. 14. This is because R3 should use more points if the IC loss is larger, but the number of IC points suggests the loss is not that large compared to the loss within the domain, resulting in a scenario that IC points are not enough for the model to learn it to a relatively low error compared to the error in the rest of the time domain. The Burgers' Equation could be an extreme case because it has a sharp change at $t = 0$(See Fig. 15 in Appendix), while the other two PDEs don't have a change as sharp as the Burgers' Equation. This is a remaining future problem in this field: how could neural networks learn a more detailed solution rather than only learn well on a global scale for the domain? Also, is it possible to simply adjust the number of points manually so that IC could be learned to a lower loss?

R3 is suitable to use in solving PDEs with smooth changes(The 1D Heat Equation and the 1D Diffusion Equation), but only two PDEs are tested in this report. PDEs with higher dimensions and larger domains, also varying hyper-parameters (such as ν in Burgers' Equation) are not tested. The result of this report only serves as a possible direction of balancing the loss between the loss within the domain, IC, and BC, but is not testing intensively to show the methods may also work in solving other PDEs.

The other method we proposed is to combine RAD with R3, which could be a good start for R3 combined with other sampling methods in solving PDEs. We only combine RAD with R3 because RAD represents the state-of-the-art sampling technique in the field of Residual-based Adaptive Resampling techniques. We put future work for using R3 together with other sampling techniques, making R3 a possible alternative to choosing numbers of points between points within the domain, points on the boundary, and points on initial time for training PINNs.

R3 is fundamentally different from the adaptive weight resampling because the adaptive weight resampling changes the coefficient related to the loss terms, while R3 uses the same coefficient for the loss terms but uses different numbers of points. We point out the fundamental difference is that R3 has the freedom to concentrate on different parts of the domain by simply using more or fewer points, while adaptive weight resampling could also explore the domain by resampling, but could not emphasize a specific part of the domain (within the domain, IC, or BC), making it not learning as careful as R3

The last question to ask is whether R3 transfers the knowledge of the initial condition and boundary condition to the solution within the domain by using more points at the beginning stage of the training (before 4000 epoch). This is a very fundamental question to ask because R3 might be learned more on the IC and BC. After all, the number of points is 2000 at the beginning of the training, and the number of points shifts to within the domain. We propose the possibility of this *knowledge transfer* in R3, but further experiments should be made to draw a firm conclusion.

6 Conclusion

In this report, we proposed a novel adaptive resampling method named R3, and we are also the first to study the number of points that could be a factor affecting the PINN training's accuracy. The R3 method compared with the current state-of-the-art resampling method RAD, achieved a lower loss for solving Burgers' Equation, Heat Equation, and Diffusion Equation with a faster convergence speed. R3 achieves either the best accuracy or a similar magnitude of loss at the end of training. R3 also shows its promising potential in balancing and reducing the loss across the time domain in solving Heat Equations and Diffusion Equations.

However, we observe that R3 is unstable: at the beginning of the training, before 10000 epochs, R3 performs better in terms of either accuracy or balancing the loss but experienced an unstable change around 10000 epochs. We also note that R3 didn't solve the Burgers' Equation with a relatively low loss $<< 10^1$ at $x = 0$. Despite that R3 and R3 with RAD achieved a smaller loss in solving the 1D-Burgers' Equation, they failed to achieve a more balanced loss across the time domain. For future work, one challenge remains is how to guide PINNs to learn better the area that has high derivatives such as the 1D Burgers' Equation at $x = 0$.

R3 started with the same number of points for collocation points, IC and BC, and later we observe the number of points for collocation points increases, while for IC and BC the number of points drops. However, we also observed that when R3 solved the diffusion equation, the best combination it gave is all points are sampled for collocation points, while IC and BC remain the minimum 100 points. This behavior is also worth studying, suggesting that balancing the number of points at the very beginning stage of training might have a positive impact on lowering the loss. Further studies are needed to test this hypothesis.

In conclusion, R3 could reach a better accuracy than the existing state-of-the-art method for at least 5% with less than 18000 epoch, while also reaching a more balanced loss across the time domain under the current experiment setting for solving Heat Equation and Diffusion Equation, but not the Burgers' Equation. We also show that RAD and RAR-D could perform worse than the baseline (Random-Uniform Resampling) under different settings, making it not the optimal resampling technique.

7 Appendix

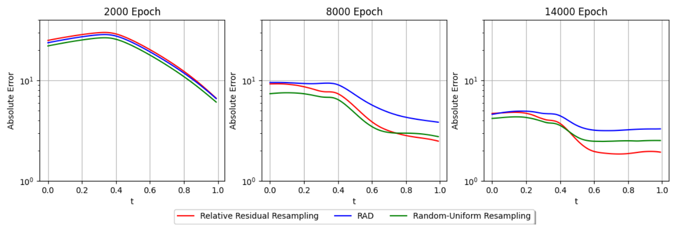
7.1 1D Allen-Cahn Equation

We set Ω to be the interval $[-1,1] \times [0,1]$ in the testing 1D Diffusion Equation.

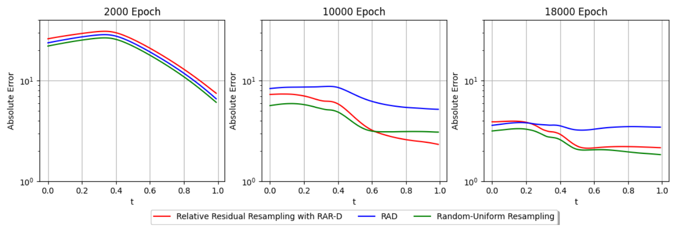
$$\frac{\partial u}{\partial t} = d\frac{\partial^2 u}{\partial x^2} - 5(u - u^3), \quad x \in [-1,1], \quad t \in [0,1],$$
$$u(x,0) = x^2 \cos(\pi x),$$
$$u(-1,t) = u(1,t) = -1. \tag{27}$$

In Eq. (27), $\frac{\partial u}{\partial t}$ represents the time derivative of u, showing how u changes over time. $d\frac{\partial^2 u}{\partial x^2}$ represents the diffusion of u in the spatial domain, where d is the diffusion efficient. $-5(u-u^3)$ is the reaction term, where 5 is a constant coefficient. This term drives the phase separation in the Allen-Cahn equation.

7.2 Experimental Results for Burgers' Equation

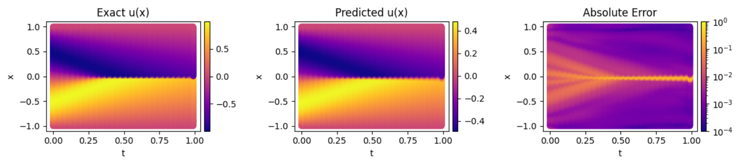


(a) R3, Random-Uniform-R and RAD

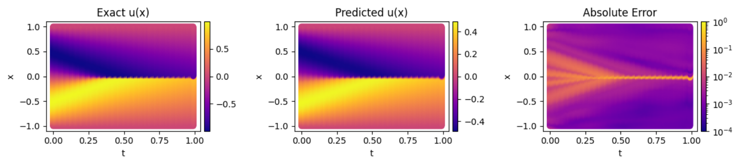


(b) R3 with RAD, Random-Uniform-R and RAD

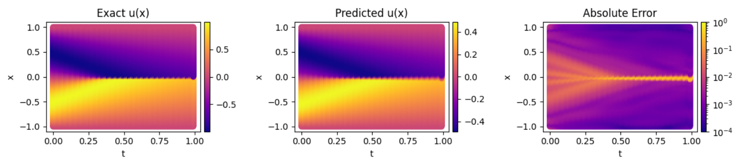
Fig. 14: One-Dimensional Burgers' Equation: (a) Comparing the loss on the time domain for the predicted solution using R3 with that of Random-Uniform Resampling and RAD. (c) Comparing the loss on the time domain for the predicted solution using R3 with RAD with that of Random-Uniform Resampling and RAD.



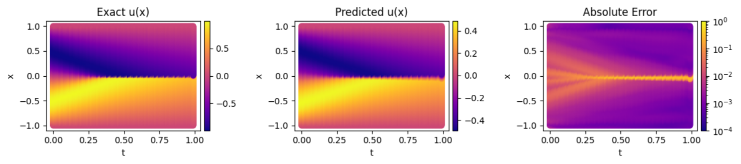
(a) R3



(b) R3 With RAD



(c) Random-Uniform Resampling



(d) RAD

Fig. 15: One-Dimensional Burgers' Equation: (a) The predicted solution versus the exact solution using R3. The relative error is 0.139 ± 0.007. (b) The predicted solution versus the exact solution using R3 with RAD. The relative error is 0.149 ± 0.001. (c) The predicted solution versus the exact solution using Random-Uniform Resampling. The relative error is 0.154 ± 0.004. (d) The predicted solution versus the exact solution using RAD. The relative error is 0.170 ± 0.01.

References

[1] 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*, vol. 378, pp. 686–707, 2019.

[2] B. Yu *et al.*, "The deep ritz method: a deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, no. 1, pp. 1–12, 2018.

[3] E. Weinan, "Machine learning and computational mathematics," *arXiv preprint arXiv:2009.14596*, 2020.

[4] A. Hrennikoff, "Solution of problems of elasticity by the framework method," 1941.

[5] R. Courant, "Variational methods for the solution of problems of equilibrium and vibrations," 1943.

[6] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[7] L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112732, 2020.

[8] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks for heat transfer problems," *Journal of Heat Transfer*, vol. 143, no. 6, p. 060801, 2021.

[9] S. Wang, X. Yu, and P. Perdikaris, "When and why pinns fail to train: A neural tangent kernel perspective," *Journal of Computational Physics*, vol. 449, p. 110768, 2022.

[10] X. Chen, "Spectrum for the allen-chan, chan-hillard, and phase-field equations for generic interfaces," *Communications in partial differential equations*, vol. 19, no. 7-8, pp. 1371–1395, 1994.

[11] S. M. Allen and J. W. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening," *Acta Metallurgica*, vol. 27, no. 6, pp. 1085–1095, 1979.

[12] C. L. Wight and J. Zhao, "Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks," *arXiv preprint arXiv:2007.04542*, 2020.

[13] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[14] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[15] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, pp. 84–90, 1960.

[16] I. M. Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, vol. 7, no. 4, pp. 784–802, 1967.

[17] J. Hammersley and D. Handscomb, "Monte carlo methods, methuen & co," *Ltd., London*, vol. 40, p. 32, 1964.

[18] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM review*, vol. 63, no. 1, pp. 208–228, 2021.

[19] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 403, p. 115671, 2023.

[20] S. Wang, S. Sankaran, and P. Perdikaris, "Respecting causality is all you need for training physics-informed neural networks," *arXiv preprint arXiv:2203.07404*, 2022.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[23] K. Tang, X. Wan, and C. Yang, "Das-pinns: A deep adaptive sampling method for solving high-dimensional partial differential equations," *Journal of Computational Physics*, vol. 476, p. 111868, 2023.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my instructor David, and my daily supervisor Mahdi. With your guidance, suggestions, and discussions, I learned a lot over the last 9 months, I couldn't do this without your help.

I also want to thank my family for their unwavering support and encouragement throughout my master's studies. Their love and faith in me have been my greatest motivation. I went back to China for the Chinese New Year, during which time I received mental support from all my family, making it an unforgettable Chinese New Year.

I would also like to thank my friends for their support during this journey. Special thanks to Alex, Chelsea, and Bogdan, with whom I have spent countless hours studying and discussing ideas. Your companionship and insights have been invaluable. I would also thank my friends from the Introduction Programme group since we were freshmen on this campus two years ago: Varun, Enrique, Deverajan, Giovanni, Jasna, Nefeli, and Jacob. I want to thank Enrique's Mom Nieves and his Dad Enrique for the emotional support when they visited us in the Netherlands, and when I visited them in Seville.

Lastly, I am immensely grateful to my girlfriend Fanshu Ma for her emotional support through all stages of the thesis, especially during the challenging final stages. Your patience, understanding, and encouragement have been my anchor, and I could not have completed this work without you.