Results in Engineering 21 (2024) 101931

Contents lists available at ScienceDirect

Results in Engineering

journal homepage: www.sciencedirect.com/journal/results-in-engineering

Full Length Article

A robust Gated-PINN to resolve local minima issues in solving differential algebraic equations

SangJoon Lee *, Byung-Tak Lee, Seok Kap Ko

Electronics and Telecommunications Research Institute (ETRI), 11, Cheomdangwagi-ro 176beon-gil, Buk-gu, Gwangju, 61012, Korea

A R T I C L E I N F O

Keywords:
Physics-informed neural network
Differential algebraic equation
Local minima problem
Robust learning

A B S T R A C T

Physics-Informed Neural Network (PINN) has emerged as a promising tool for solving various physical problems with differential equations. However in practice, PINN often suffers from the local minima issue while solving problems with minimal initial conditions. In this paper, we propose a robust Gated-PINN to address this issue, which blends numerical methods and PINN. The Gated-PINN overcomes the local minima issue by controlling the flow of information of the neural network similar to a numerical method. We first investigate the use of conventional PINN to the Cartesian-coordinate simple pendulum problem with minimal initial conditions, as one of basic differential algebraic equation (DAE) problem. Our results show the advantage of PINN over existing numerical methods in that it does not require sophisticated mathematical techniques such as index reduction. But we also observe that conventional PINN can lead to inaccurate solutions; such solutions partially satisfy the differential equation requirement, but does not meet the given initial condition and fail to further improve. We demonstrate the effectiveness of the proposed Gated-PINN by showing that it yields accurate solutions, such that for a pendulum of length 1, the mean Euclidean error between the Gated-PINN model and the traditional numerical method model is less than 0.01 and pointwise maximum Euclidean error is less than 0.04. Moreover, Gated-PINN can operate without any complicated index reduction, and unlike conventional PINN, accurate solution can be obtained consistently without falling into a local minima. Overall, our study presents the potential of the Gated-PINN for solving DAE problems and provides a valuable insight into the challenges and limitations of using PINN for solving physical problems.

1. Introduction: PINN and DAE

Differential equations are ubiquitous in physics and engineering, and their solutions often play a critical role in understanding the underlying physical phenomena. Recently, physics-informed neural networks (PINN) have emerged as a powerful tool for solving differential equations. PINN is a deep learning-based approach that combines the flexibility of neural networks with the physical laws that govern the system being modeled [24]. Unlike traditional numerical methods, PINN does not require a grid or mesh to discretize the domain, making it particularly well-suited for problems with complex geometries. PINN has shown great promise in solving various physical problems, such as hyperelasticity problems [23], inverse problems [8,19], thermal modeling [3,5], power systems [20], civil engineering [29] and fluid dynamics [28,15–17,30,25,9,26,32,10,31,13,4,14,6].

While the application of PINN has mainly been focused on the problems involving PDEs, the general idea of PINN can be applied to a much wider variety of problems with some mathematical structure embedded inside. In particular, we put our attention on differential algebraic equation (DAE). A DAE is a mixed system of differential and algebraic equations with at least one unknown variable x that is purely algebraic. DAE usually refers to problems with only one independent variable; problems with more than one variable which partial derivatives are taken with respect to are referred as partial differential algebraic equation (PDAE).

In general, DAEs are more challenging to solve than ODE. This is due to an algebraic variable x in a DAE. Because x does not have a differential relation, we cannot solve for dx/dt through algebraic manipulation, thus the typical numerical step process cannot be done on x. As a consequence, many existing ODE solvers such as MATLAB often

* Corresponding author.
E-mail addresses: sjlee85@etri.re.kr (S. Lee), bytelee@etri.re.kr (B.-T. Lee), softgear@etri.re.kr (S. Kap Ko).

https://doi.org/10.1016/j.rineng.2024.101931
Received 23 November 2023; Received in revised form 1 February 2024; Accepted 17 February 2024
Available online 23 February 2024

cannot solve a given DAE by its raw form; the user needs to manually reduce the index of a DAE to make it feasible for the solver.

One of the main obstacles in solving a DAE is that this index reduction process is highly problem dependent, but is also essential for numerical solving. This makes DAEs difficult to utilize for practical applications. PINN on the other hand does not solve the problem by derivatives, therefore it may not require derivatives of each specific variable. This makes PINN an attractive candidate for index reduction-free DAE solver. Recently there has been notable development on this problem regarding power network-related DAEs [22], which the authors utilized the Runge-Kutta time-stepping scheme-based PINN to solve the problem. In this paper we study the performance of a conventional PINN, which has the advantage of straightforward implementation and mesh-free nature, and the drawback of being susceptible to stiffness in general.

In section 2, we provide the mathematical description of the Cartesian simple pendulum problem, and discuss the procedure of solving it numerically. In section 3, we provide how the pendulum problem was solved using a conventional PINN, and discuss why we did not get a good result. In section 4, we introduce the idea of our Gated-PINN, and apply it to the pendulum problem. We present a detailed assessment of Gated-PINN in section 5.

2. The simple pendulum problem

A simple pendulum problem is typically described as an ODE for the angle θ of the pendulum against the vertical axis. However, we can also describe this problem in a Cartesian coordinate system, where the pivot of the pendulum is defined as the origin of the coordinate system. This yields the following system of equations.

$$
\begin{aligned}
\ddot{x}(t) &= \lambda(t)x(t) \\
\ddot{y}(t) &= \lambda(t)y(t) - g \\
x(t)^2 + y(t)^2 &= L^2
\end{aligned} \tag{1}
$$

$x(t)$ is the x-coordinate of the pendulum at time t, $y(t)$ is the y-coordinate of the pendulum at time t, $\lambda(t)$ is a Lagrange multiplier, and L and g are constants representing the length of the pendulum and the gravitational acceleration respectively. The second derivatives of x and y make this equation second-order, and since λ does not have a differential component, this is a DAE problem. More specifically, this is an index-3 DAE problem, that is, we need to differentiate and manipulate the components of this equation three times to transform this problem into an ODE problem. For this example, differentiating the last equation gives the following

$$\dot{x}x + \dot{y}y = 0 \tag{2}$$

which makes the problem index-2. After another differentiation and some algebraic manipulation, we get

$$L^2\lambda - gy + \dot{x}^2 + \dot{y}^2 = 0 \tag{3}$$

which makes the problem index-1, and one more step of a similar procedure gives us

$$L^2\dot{\lambda} - 3g\dot{y} = 0 \tag{4}$$

which finally gives us some information on the derivative of λ. Standard numerical DAE solvers often require the problem to be index-1, so along with order reduction, (1) needs to be written as follows:

$$
\begin{gathered}
\dot{x}(t) = u(t) \qquad \dot{y}(t) = v(t) \\
\dot{u}(t) = \lambda(t)x(t) \qquad \dot{v}(t) = \lambda(t)y(t) - g \\
L^2\lambda - gy + u^2 + v^2 = 0
\end{gathered} \tag{5}
$$

Fig. 1 illustrates the problem and shows the plot of $x(t), y(t), \dot{x}(t)$ and $\dot{y}(t)$ numerically solved by MATLAB ode15 solver [27] using (5) with $L = g = 1$, $y(0) = \dot{x}(0) = \dot{y}(0) = 0$ and $x(0) = 1$.

It turns out that the PINN approach on the pendulum problem is both order and index invariant, that is, it can sufficiently approximate the motion of the pendulum under any of the order-index version of the problem. Fig. 2 shows that PINN is capable of solving the simple pendulum regardless of the order or index of the formulation.

This result suggests that PINN does not require any nontrivial procedure to solve a given DAE problem. Granted, it does require some procedures such as input-output normalization, but such a process can be standardized [11]. While this observation shows that PINN can be an attractive tool to solve general DAE problems, we have also observed some of the fundamental downside of PINN as a member of the machine learning family; the local minima issue. In the following sections, we will discuss how and why this problem occurs, and present our novel method of resolving it.

3. Local minima issue on conventional PINN

3.1. PINN training failure on pendulum problem

The results in Fig. 2 were obtained using a traditional PINN with standard multi-layer perceptron (MLP) structure. Specifically, we have a single data point at $t = 0$:

$$x(0) = 1, \quad y(0) = \dot{x}(0) = \dot{y}(0) = \lambda(0) = 0 \tag{6}$$

and we want to forecast the motion of the pendulum for $0 \le t \le 10$.

PINN using the order-1 version of DAE consists of an MLP with one-input (t) and five (x, y, u, v, λ) outputs, with 8 hidden layers having 20 neurons each.

The loss function is split into initial loss and collocation loss:

$$
\begin{gathered}
L_{x_0} = (x_\theta(0) - x(0))^2 \qquad L_{y_0} = (y_\theta(0) - y(0))^2 \\
L_{u_0} = (u_\theta(0) - u(0))^2 \qquad L_{v_0} = (v_\theta(0) - v(0))^2 \\
L_{\lambda_0} = (\lambda_\theta(0) - \lambda(0))^2 \\
L_{init} = L_{x_0} + L_{y_0} + L_{u_0} + L_{v_0} + L_{\lambda_0}
\end{gathered} \tag{7}
$$

$$
\begin{gathered}
L_1 = \sum_{i=1}^{N}(\dot{x_\theta}(t_i) - u_\theta(t_i))^2 \qquad L_2 = \sum_{i=1}^{N}(\dot{y_\theta}(t_i) - v_\theta(t_i))^2 \\
L_3 = \sum_{i=1}^{N}(\dot{u_\theta}(t_i) - \lambda(t_i)x(t_i))^2 \qquad L_4 = \sum_{i=1}^{N}(\dot{v_\theta}(t_i) - \lambda(t_i)y(t_i) + g)^2 \\
L_5 = \sum_{i=1}^{N}(L^2\lambda(t_i) - gy(t_i) + u(t_i)^2 + v(t_i)^2)^2 \\
L_{colloc} = \frac{L_1 + L_2 + L_3 + L_4 + L_5}{N}
\end{gathered} \tag{8}
$$

Collocation loss is computed over $N = 1{,}000$ collocation points (t_i), uniformly sampled from $0 \le t \le 10$. Each term in L_{colloc} corresponds to the DAE components described in (5). L_5 in particular is specific to the index-1 version of the problem; it can be replaced with (2) or $x^2 + y^2 = L$ for index-2 and index-3 respectively.

PINN using the order-2 version of DAE consists of a similar MLP but only has three outputs (x, y, λ). The loss function is also simplified accordingly.

$$
\begin{gathered}
L_{x_0} = (x_\theta(0) - x(0))^2 \qquad L_{y_0} = (y_\theta(0) - y(0))^2 \\
L_{\lambda_0} = (\lambda_\theta(0) - \lambda(0))^2 \qquad L_{init} = L_{x_0} + L_{y_0} + L_{\lambda_0}
\end{gathered} \tag{9}
$$

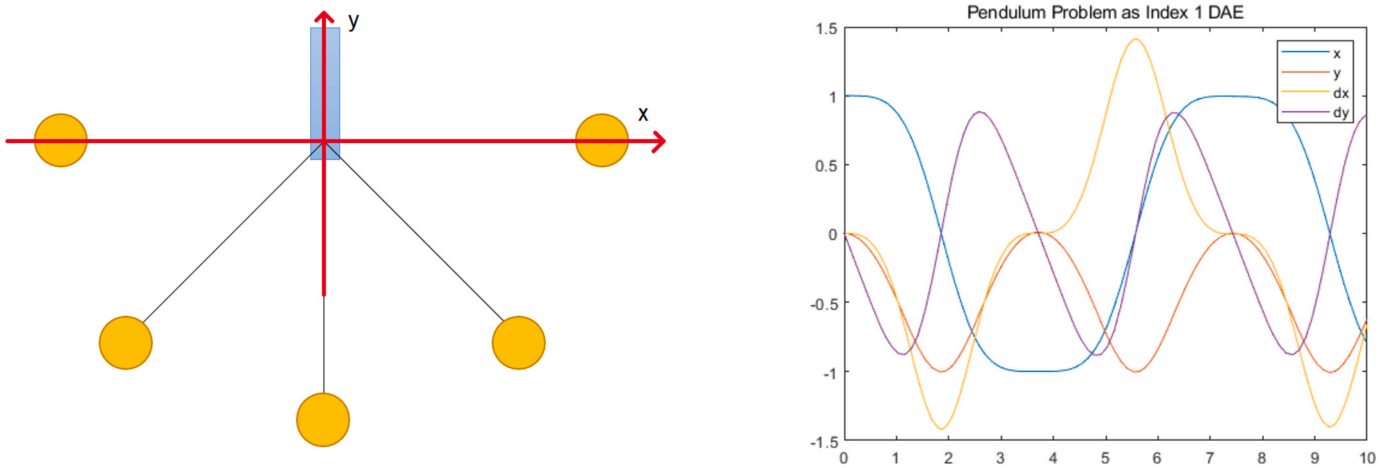

Fig. 1. Left: A simple pendulum described in Cartesian coordinates. Right: MATLAB ode15 solution of the simple pendulum.

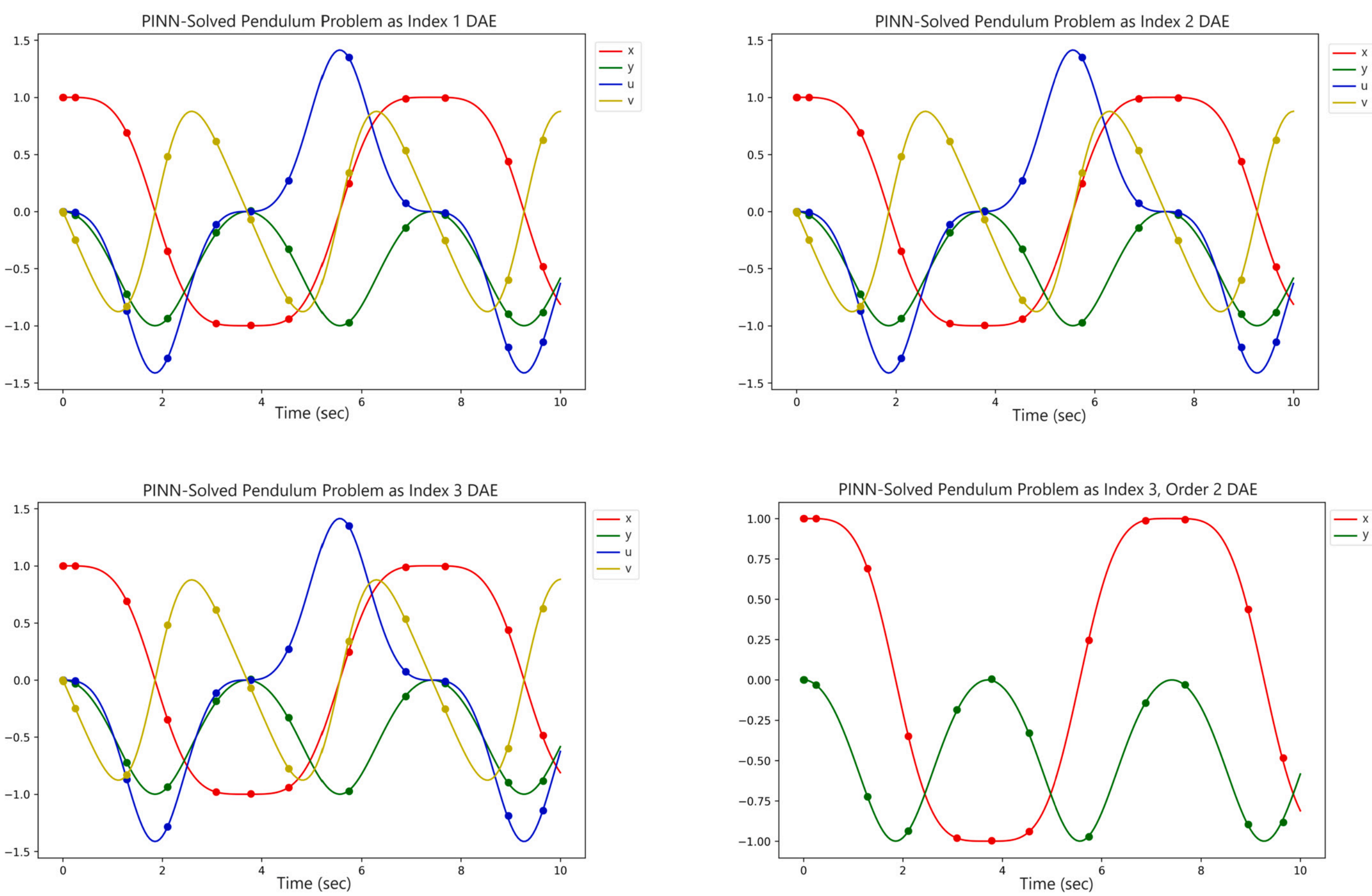

Fig. 2. PINN solution of the Cartesian simple pendulum problem using different index and order version formula. Dotted points indicate the reference values obtained from MATLAB solution. Order-2 formula (bottom right) is the most natural formulation (1), and unlike order-1 formula it only yields $x(t)$ and $y(t)$ of the pendulum directly.

$$L_1 = \sum_{i=1}^{N} (\ddot{x}_\theta(t_i) - \lambda(t_i)x(t_i))^2 \qquad L_2 = \sum_{i=1}^{N} (\ddot{y}_\theta(t_i) - \lambda(t_i)y(t_i) + g)^2$$
$$L_3 = \sum_{i=1}^{N} (x_\theta(t_i)^2 + y_\theta(t_i)^2 - L^2)^2 \qquad L_{colloc} = \frac{L_1 + L_2 + L_3}{N} \tag{10}$$

Like before, L_3 is specific to the index-3 version of the problem. Throughout the rest of the paper, unless otherwise specified, our PINN structure will use a standard-weight loss function $L = L_{init} + L_{colloc}$, and Adam optimizer with learning rate of $1e-3$.

Fig. 2 shows that conventional PINN can solve the pendulum using any of the loss functions described above. However, it turns out that the training is not robust, even though the data is created noise-free. Fig. 3 shows different types of failed PINN training. The loss evolution graph shows that the model tries to find some convergence, but after many epochs the model is still far from the correct solution.

Each failed model results in the trained values stable for large t. This suggests that PINN guesses the pendulum is at a steady state after some time; in fact, our pendulum does have two steady state solutions

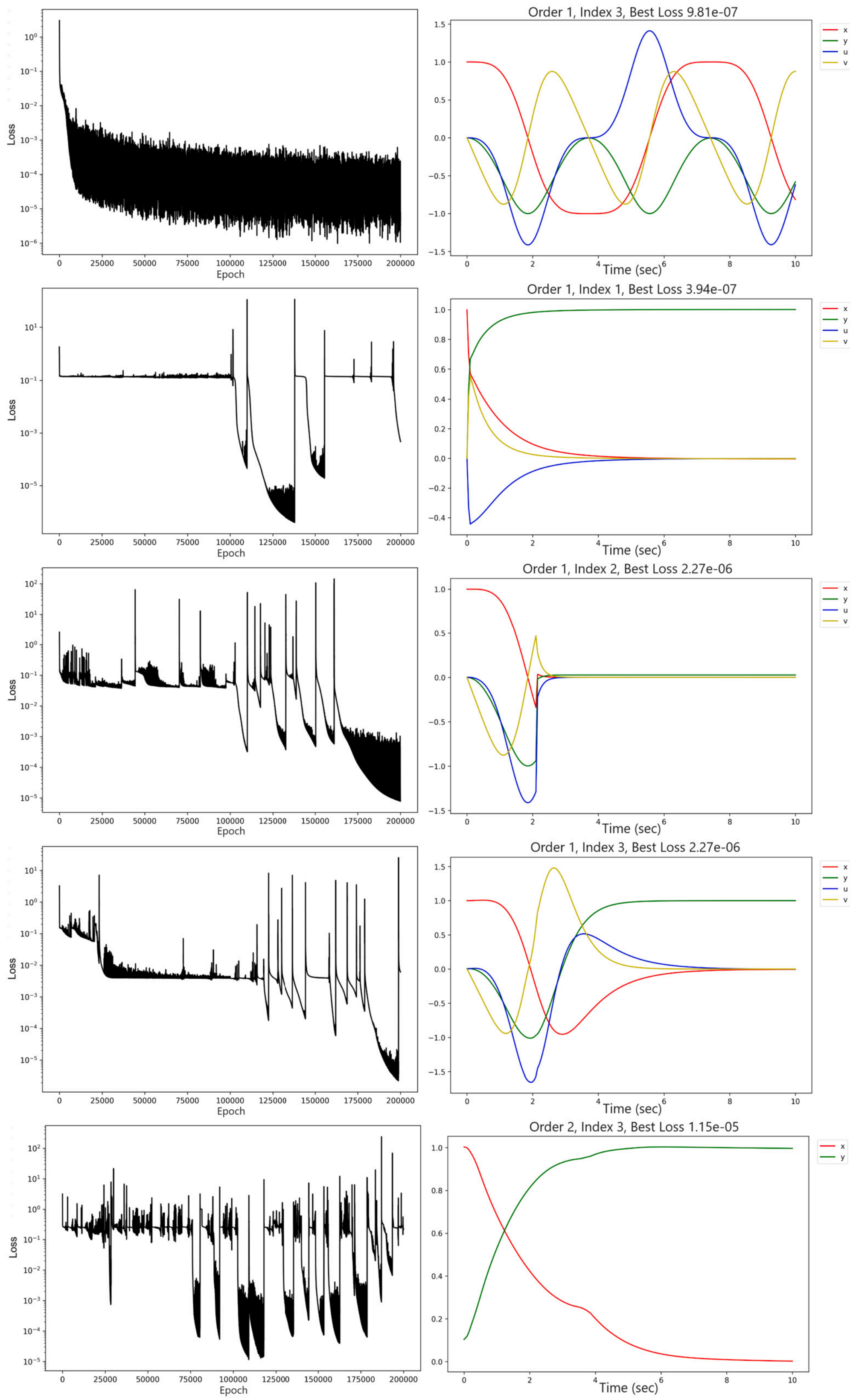

Fig. 3. Loss evolution and trained pendulum model under different order-index settings. Other than the first experiment, conventional PINN struggles to escape from local minima.

at $(x, y) = (0, \pm 1)$. However, this convergence should not happen since our model did not start from such state and is friction-free.

Also, this training failure happens quite often; we ran minimum 36 runs of conventional PINN training on different fixed Numpy and Tensorflow random seeds, across different order and index version of DAE loss function for the PINN structure. For each version, our training yields little more than 50% case of successful model, and no version was able to produce more than 70% case of successful model. This experiment suggests that the successfulness of pendulum training with conventional PINN is dependent on the random seed and its related random number generation, and this issue is independent from either order or index of the problem.

We believe each of these failed training is the result of converging to a local minima, and want to avoid this issue. Maintaining the minimal requirement is the key concept here; if we relax this condition, our problem can be easily solved by adding extra data points for larger t. From our experiment, adding the condition $y(t) < 0$ also lead to fully successful training; despite the existence of another steady state solution at $(x, y) = (0, -1)$, our model always yields the correct periodic solution with this condition. However this too would be a forced extra condition that may cause problems for different initial conditions. Our goal is to develop a modified PINN structure that is robust enough to consistently solve the DAE without the index and order reductions, while not adding any extra restrictions other than the essential condition a numerical method would require. Also, it is worth noting that such a training issue is not limited to just DAEs; there have been studies that shows conventional PINN training to struggle in different classes of problems, and while many alternate methods have been proposed, their performance varies over different problems [21].

3.2. Comparing numerical method and conventional PINN

It is important that we try to understand the difference between numerical methods and PINN to find the solution to a given DAE, or any initial value problem in general. For a numerical method, the estimated function value at each point is dependent on the function value at the previous point. Combined with the fact that numerical methods search the solution recursively, the given initial condition of the problem is guaranteed to have full influence throughout the domain.

PINN on the other hand work differently; it tries to find the function value over the entire domain simultaneously, while checking the DAE restrictions at each collocation point. Unlike numerical method, the initial condition (or any reference data point for that matter) does not have a direct impact on estimating function values at future points, and its influence diminishes further on. Due to these characteristics, at the early stages of epochs and for large values of t, PINN will fit the model to a stable state. And as the epoch advances, the model will trap itself further into a local minima, and the initial condition will not have strong enough influence to dig the model out from it.

One idea to deal with this issue would be weight adjustment between initial loss (9) and collocation loss (10). By giving higher weight on L_{init}, we can emphasize the importance of the initial condition and hope that it will carry further into the time domain. But there are two problems; in practice, this idea would make the model vulnerable to noise in data measurement [1], and more importantly, even with the emphasis on the initial condition, its influence seems to quickly disappear as time domain advances. We ran experiments with this idea of weight adjustment, and the results suggested that it can only temporarily drive the model to the right direction, but not throughout the time domain.

Although the weight adjustment did not solve our issue, it did give us an idea; if f is the solution to a given initial value problem and $f(t)$ is to be estimated by PINN, $f(s)$ for $s < t$ must have sufficient influence on training $f(t)$ while not letting the vice versa to happen. In the next section, we represent Gated-PINN, our new PINN model to implement this idea.

4. Proposal of Gated-PINN

4.1. Idea and algorithm

Suppose $F_\theta(t) = [x_\theta(t), y_\theta(t), \lambda_\theta(t)]$ is our PINN for training the pendulum model (1). If $F(t)$ is the true solution, we have so far observed the following two properties.

1. If $F(t_0)$ is given as a known data point, $F_\theta(t)$ trains well in the neighborhood of t_0.
2. If F_θ is initialized close enough to F[1], PINN is more likely to converge to the correct global minima.

The local minima issue described in section 3 occurs when F_θ initially fits to a piecewise blend of the correct solution near $t = 0$ and an incorrect steady-state solution far from $t = 0$. This causes the function distance $d(F_\theta, F)$ to be too large for PINN to correctly converge.

Our strategy is to utilize this piecewise characteristic. If we fix $t_1 < 10$ and define a temporary domain $T_1 = [0, t_1]$ that is a subset of our full domain $T = [0, 10]$, the function distance $d(\,F_\theta\big|_{T_1}, F\big|_{T_1})$ would be smaller than $d(F_\theta, F)$, even after normalizing it by the length of respective domain. If t_1 is sufficiently small, PINN can accurately approximate F_θ by the first property. Also, if F_θ is well-trained on T_i, it can serve as a good initialization for training F_θ on a slightly increased domain T_{i+1}, so F_θ can be accurately extended to T_{i+1} by the second property. Finally by continuing this process, PINN can recursively train F_θ until it is trained on the full domain T.

The procedure of our Gated-PINN is further described in Algorithm 1. Fig. 4 illustrates a rough sketch of how Gated-PINN would train a Cartesian simple pendulum model.

Algorithm 1 Gated-PINN for DAE

DAE time domain $T = [0, t_{max}]$, DAE equations $\mathcal{F}$, Initial condition I
G-PINN surrogate MLP F_θ, Set of Collocation Points R, PINN Hyperparameter Set H
Initial domain indicator t_0, Phase Epoch sequence N_1, N_2, N_3
$T_1 \leftarrow [0, t_0]$
for $i = 1$ to $N_1 + N_2 + N_3$ **do**
  $L_{init} \leftarrow$ Initial condition error on I
  $L_{colloc} \leftarrow$ DAE collocation error on $\mathcal{F}, R$
  if $i \le N_1$ **then** ▷ Phase 1
    $L \leftarrow SSE(L_{init}) + MSE(\mathbf{1}_{T_1} \cdot L_{colloc})$
  else if $i \le N_1 + N_2$ **then** ▷ Phase 2
    $T_i \leftarrow [0, t_0 + (t_{max} - t_0) * (i - N_1)/N_2]$
    $L \leftarrow SSE(L_{init}) + MSE(\mathbf{1}_{T_i} \cdot L_{colloc})$
  else ▷ Phase 3
    $L \leftarrow SSE(L_{init}) + MSE(L_{colloc})$
  end if
  Update F_θ
end for

We add some comment on the Gated-PINN.

1. Gated-PINN works similarly to numerical methods, in a sense that the estimation of $F(t)$ occurs numerically.
2. Unlike numerical methods in which the error accumulates as we propagate along the domain, Gated-PINN continues to fit F_θ simultaneously over the domain of interest, so the estimation error can be retrospectively corrected.
3. The initial domain T_1 need not be too small. In fact, a larger T_1 (that can consistently find accurate $F_\theta\big|_{T_1}$) yields a better initial surrogate for the rest of the training process.

[1] By close enough, we mean the L^p distance between functions being small, but for our purpose the criterion need not be so rigorous and we will not go into further details.

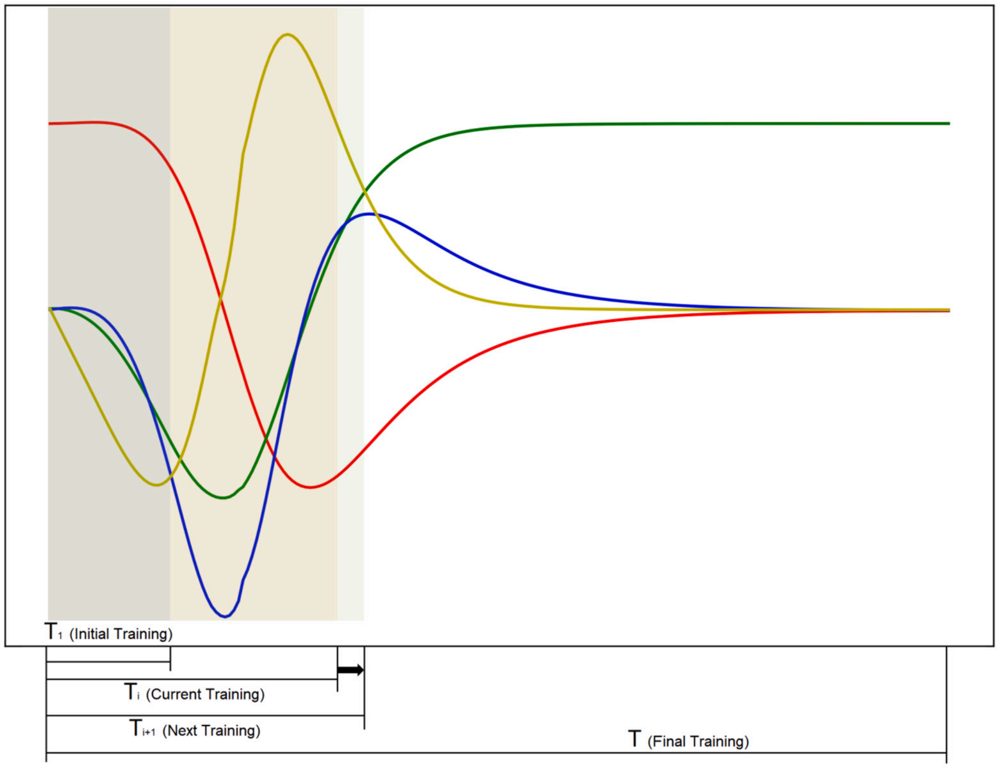

Fig. 4. Graphical idea of Gated-PINN training process. The gate initially opens for T_1, then gradually widens as PINN focuses training the model for the currently open domain.

4. During the second phase, when F_θ trains on a restricted domain T_i, the process would likely affect the function value outside of T_i. So to preserve the already trained portion of F_θ, it is essential that $T_i \subset T_{i+1}$.
5. We used linear factor $(i - N_1)/N_2$ to increase training domain for the second phase; a concave increasing factor such as $\sqrt{(i - N_1)/N_2}$ can be used instead to allow longer training time for larger domain.
6. N_3 is the number of epoch which F_θ trains similarly to a conventional PINN. Some conventional MLP hyperparameters may be adjusted specifically for this stage.

4.2. Related works

The idea of applying numerical time-step principle on PINN has been studied before. Backward compatible PINN (bc-PINN) [18] initially runs PINN only on the early portion of the time domain, samples some points from the trained domain, then trains on the next portion from the transferred model, using the previously sampled points as data guide. One possible issue with bc-PINN is that at any stage of training, bc-PINN only focuses on the governing physics constraint at the particular portion of the domain; although the newly sampled points serve as a guide to preserve the training on previous sections, the physics structure on those sections is ignored and could be distorted during the latter stage of training.

The improved backward compatible PINN (ibc-PINN) [12] addresses this issue by keeping a separate neural network model on each partial section of PINN training. They pointed out that each sub-model is likely to be most accurate on the particular domain where the governing physics information was considered, so each intermediate guide data sampling comes from its corresponding sub-model. The final trained model is also represented as piecewise combination of partially trained sub-models. While ibc-PINN clearly has an advantage over bc-PINN in training accuracy, the drawback of this method is that the overall model size grows proportionally as the number of time subsections increases, so it is difficult to divide the time domain into too many subsections.

Unlike the above two methods which split the time domain into disjoint intervals, the pre-training PINN (PT-PINN) [7] approach is similar to our G-PINN approach as it use an expanding time interval. The initial training and point sampling are similar to those of bc-PINN, but unlike bc-PINN, the governing physics constraints monitored at the early time interval are continuously monitored throughout the latter part of training process. PT-PINN also uses a resampling strategy on the collocation points to avoid local minima.

The fundamental difference between these works and G-PINN is the goal of its design. The above three methods, as well as other modified PINN methods primarily focus on the solvability, accuracy, and training efficiency. Our work instead focuses on the robustness of the method. While we develop a time-step modified method from conventional PINN, we avoid any modification that could potentially pull the model away from the correct training trajectory, which is why G-PINN uses an expanding time interval instead of disjoint time interval for collocation error monitoring. Also, G-PINN does not sample any data points for supervision; although accurate supervision data points greatly stabilize the robustness of the training, the risk of generating bad supervision data points from premature pre-training cannot be ignored. It is also worth mentioning that fixing supervision data points makes the PINN structure unable to retrospectively correct the estimation error at a later stage; either a resampling strategy (as utilized in PT-PINN) or no additional sampling at all (as utilized in G-PINN) is needed to solve this issue. Finally, each of the above methods takes a relatively large discrete time step for implementation; if $[0, T]$ is the time domain of the interest, the time sequence $0 = t_0, t_1, \dots t_k = T$ is set such that $k < 10$. G-PINN, in contrast, as described in Algorithm 1, expands the time domain linearly from $[0, t_i]$ and the maximum index k is in thousands. This is due to our observation that when a large chunk of new collocation points is introduced at once, the training process becomes unstable and could fail. Gradually adding small number of new collocation points can

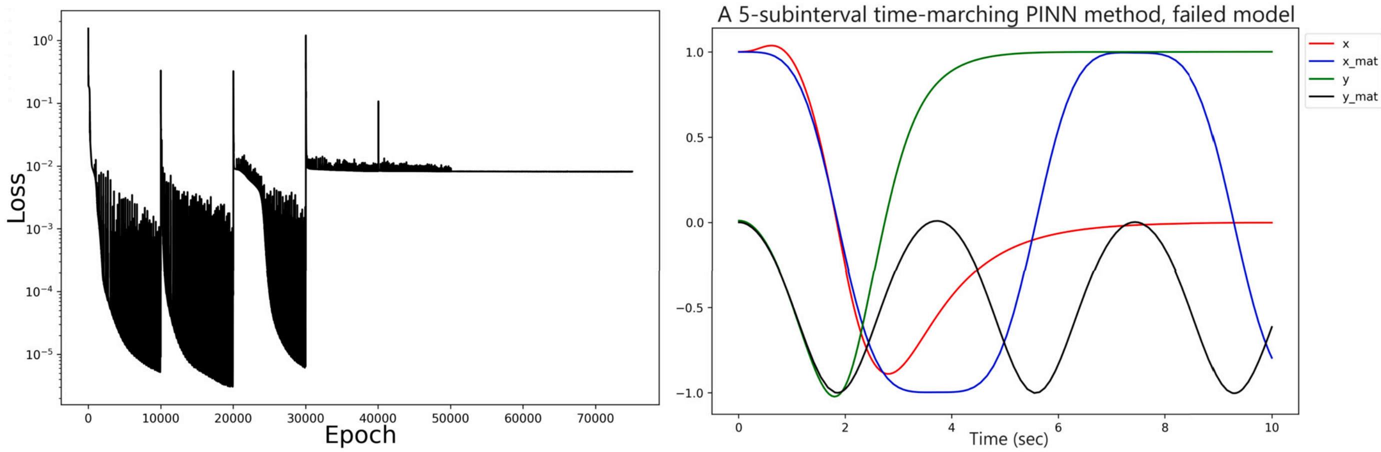

Fig. 5. A failed pendulum model with time-marching PINN training with 5 subintervals. The model was initially trained on time interval $[0,2]$, then subsequently expanded to $[0,4]$, $[0,6]$, $[0,8]$ and $[0,10]$. The loss evolution graph suggests that adding large chunk of interval makes the training temporarily unstable, and by the expansion to $[0,8]$ the model falls into local minima. $x$mat and $y$mat are the reference MATLAB solutions.

relatively stabilize the training process. The training stability difference between small and large values of k is displayed in Fig. 5 and Fig. 6.

4.3. Gated-PINN on Cartesian pendulum problem

We used Gated-PINN on the order-2 index-3 simple pendulum model (1), with 36 different random seeds tested on. The overall MLP structure is the same as the one used in section 3.1. Unlike conventional PINN, we were able to get successful training on each of the random seeds. When initial domain T_1 is set to $[0,2]$, the initialized model after training on T_1 shows different tail behavior for different seeds. Despite that, Gated-PINN is able to train each case to the desired global minima. As for the training time, because we do not update the set of collocation points R along the progression of T_i and instead use a masking factor 1_{T_i} to limit the contribution of L_{colloc}, the training time across all phases is consistent per epoch. Using Nvidia Tesla V100 GPU, Tensorflow 2.10 and Float32 data type, the training time is approximately 0.0082 seconds per epoch. We comment that updating the set of collocation points itself (update R_i as T_i progress) may improve the overall computation time, as the earlier stage of training would account for a smaller number of collocation points to compute.

Fig. 6 shows the initial, intermediate, and final pendulum models trained by G-PINN, the loss evolution, and the comparison with the FDM MATLAB solution. As expected, the absolute difference between the two methods increases as t increases. The pointwise difference pattern can be understood from how the G-PINN model evolves over phase 3 of the training, where the period of the pattern was initially overestimated and slowly decrease to the correct period. Also, the loss evolution suggests that G-PINN model was still under the converging process, therefore it can be easily further improved by further training and learning rate adjustment.

5. Assessment of Gated-PINN

To further investigate the performance of G-PINN, we implemented Algorithm 1 with two hyperparameters for G-PINN. N is the training epochs coefficient, such that $N_1 + N_2 = 10 \cdot N$ and $N_3 = 5 \cdot N$, so there will be $15 \cdot N$ overall training epochs. p is the coefficient that splits phase 1 and 2. p portion of $10N$ epochs will be on phase 1, which will train our model on the initial portion T_1, the first p portion of the whole domain. For the remaining $(1-p)$ portion of the $10N$ epochs, T_i will uniformly increase from T_1 to T by each epoch.

For example, on our pendulum problem, if $N = 1000$ and $p = 0.1$, we first train the model for 1000 epochs on the domain $T_1 = [0,1]$. The training continues with expanding domain $T_{i+1} = [0, t_i + 0.001]$ for the next 9000 epochs, and the model will go through final phase of training on $T = [0,10]$ for 5000 epochs.

To define a successfully trained model, we compare the model F_θ yielded by G-PINN and the model F_M computed from MATLAB. As shown in Fig. 6, we take the absolute point-wise difference, and choose the maximum value as our error.

$$Err = \max_{x,y} \left(\max_{t \in T} |F_\theta(t) - F_M(t)| \right) \tag{11}$$

If $Err < 0.1$, we accept the model as a successfully trained model. At such a range, the graphs of F_θ and F_M mostly agree, and F_θ is unlikely to bounce out to an incorrect local minima. We also comment that we mostly get $Err < 0.05$ for successful models, and the model can be additionally trained for further accuracy, but Err value will generally saturate at around 0.03.

For each pair of N and p, we train G-PINN for 36 different random seeds and count the number of successful trainings. Also, to see the effect of learning rate, we run the whole set of experiment under three different learning rate schemes; for phase 1 and 2, we train our model with learning rate $1e-2, 1e-3, 1e-4$ respectively, and for phase 3 we reduce the learning rate to $1e-3, 1e-4, 1e-5$ respectively.

Fig. 7 shows the heatmap of our experimental results. Under the right learning rate, G-PINN can consistently train the correct model as long as T_1 is less than half of T and we run a sufficiently many numbers of epochs (75,000 total). When p is close to 1, the result is similar to learning on a conventional PINN; models under certain random seeds converge to the correct model fast, but models with other random seeds fall into local minima and do not recover.

We add some discussions regarding traditional MLP hyperparameters and order/index version of this experiment.

1. The effect of learning rate for G-PINN is similar to that of a traditional deep learning MLP. If the rate is too high, the model does not converge. If the rate is too low, the learning process takes much longer to train or can fall into a local minima. Fig. 7 shows that the $1e-3 \to 1e-4$ learning rate setting is the optimal setting; for the $1e-4 \to 1e-5$ setting, some models take longer training epochs ($N > 8000$) to converge and some models even fail to escape from a local minima.
2. Adding more layers or node per layer does not affect the training robustness or model accuracy. We have conducted similar experiment with 12 layers instead of 8, as well as 30 nodes per layer instead of 20, particularly with $p = 0.2$ and $N = 6000$. For both cases, the successful training rate was 100%, but the average Err value (0.0319, 0.0317 respectively) was within the standard devia-

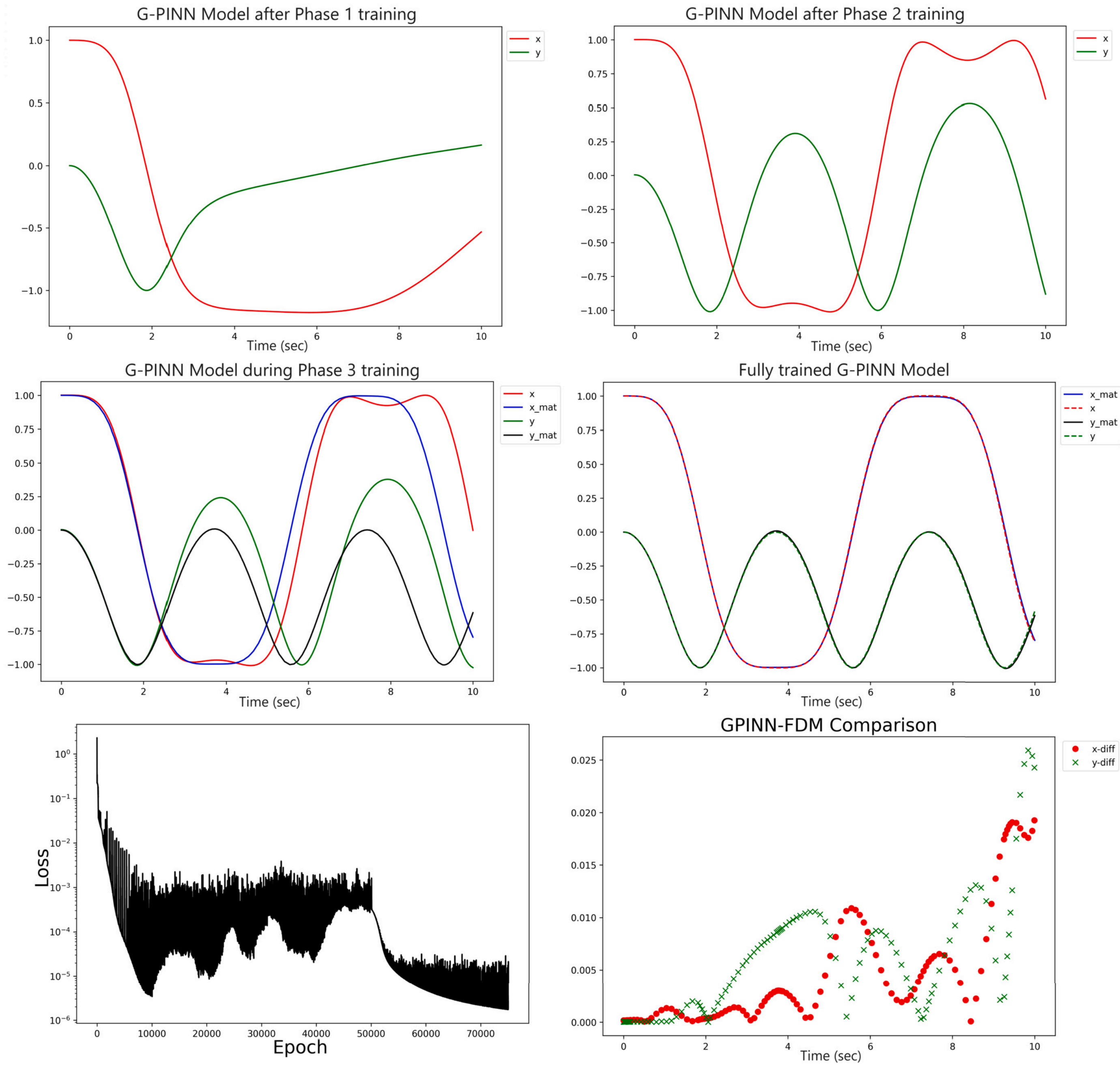

Fig. 6. Top: Pendulum model F_θ after the first two phases of training respectively. Middle: Training procedure of third phase of G-PINN along with MATLAB solution. Bottom Left: Evolution of loss value. $N_1 = 10,000$, $N_2 = 50,000$ and learning rate $lr = 1e-3$ for the first two phases, $N_3 = 25,000$ and $lr = 1e-4$ for the third phase. Bottom Right: Absolute difference between $F_\theta(t)$ obtained by G-PINN and $F_M(t)$ obtained by MATLAB.

tion (0.0027) of the base model (8 layers, 20 nodes each) results (0.0310). Due to the expanded MLP structure, the training time is 0.0139 seconds per epochs for the 12 layer model, and 0.0105 seconds per epochs for the 30 nodes-per-layer model.

3. We can apply G-PINN to the order-1 version of the pendulum problem. Similar experiments show that order-1, index-3 version takes more training epochs ($N > 8000$) to consistently converge, but the average Err value among successfully trained model was slightly better (0.0281) compared to the order-2 version. This could be due to the fact that the PINN training of the order-1 version only requires first-order automatic differentiation, which makes the overall model less prone to computation error [2]. The index-2 version model also converges consistently but takes even more training epochs ($N > 12000$), and has a worse average Err value (0.0417).
4. The index-1 version model in particular struggles to consistently converge even with $N > 20000$. We do not have a clear explanation to this; one possible mathematical reasoning could be the fact that it takes differentiation and numerous algebraic manipulations for index reduction, and the transition from a higher index equation to a lower index equation might have lost some crucial information during the differentiation process.

6. Conclusion

One of the values we expected from PINN in solving differential equations and general mathematical problems is how it approaches the problem differently from existing numerical methods. We were able to observe such a characteristic by solving a DAE without any preprocess

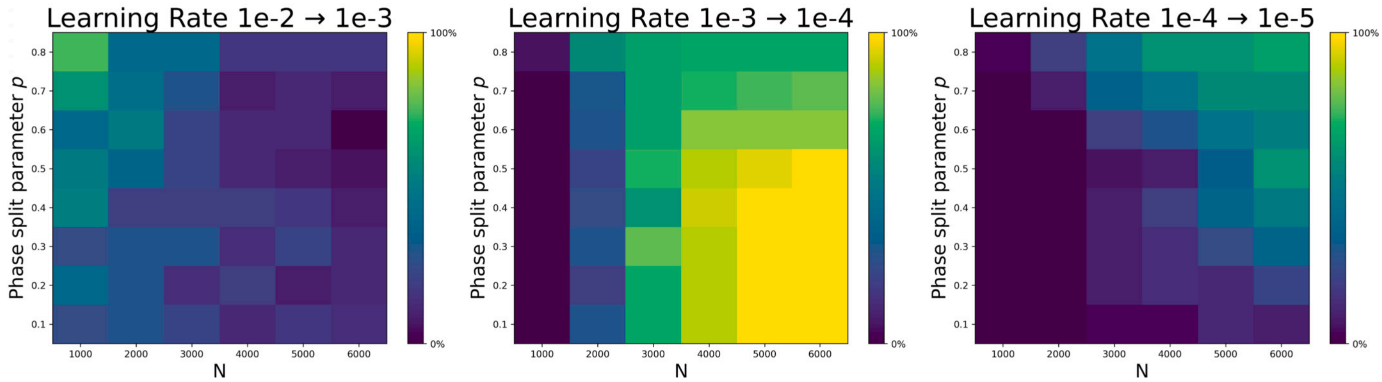

Fig. 7. Heatmap of successful G-PINN training probability by learning rate, training speed and initial gate portion.

of index or order reduction. However, a conventional PINN also has its limitation of not being able to consistently train a correct model, even for a fairly elementary problem such as a simple pendulum.

Similar to the early development of deep neural networks, we need modifications to the PINN structure in order to obtain robust applicability. We believe that our Gated-PINN can serve as a fundamental step in that direction.

Although this paper focuses on problems with a one-dimensional domain, we think our idea of a training method can be extended to PINN on a multi-dimensional domain by increasing the training domain from data validation points near the boundary to those further away. In the future, we aim to implement this idea for relevant partial differential equations.

CRediT authorship contribution statement

SangJoon Lee: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Byung-Tak Lee:** Supervision, Project administration, Methodology, Conceptualization. **Seok Kap Ko:** Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: SangJoon Lee reports financial support was provided by Government of the Republic of Korea. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work was supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government [23ZK1100, Honam region regional industry-based ICT convergence technology advancement support project]. The authors would like to express their deep gratitude to an anonymous referee whose input was invaluable to the presentation of our results and helped correcting errors and omissions. The authors would also like to thank research group members for constructive discussion and support.

References

[1] C. Bajaj, L. McLennan, T. Andeen, A. Roy, Robust learning of physics informed neural networks, arXiv preprint, arXiv:2110.13330, 2021.

[2] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.

[3] F. Bragone, K. Morozovska, P. Hilber, T. Laneryd, M. Luvisotto, Physics-informed neural networks for modelling power transformer's dynamic thermal behaviour, Electr. Power Syst. Res. 211 (2022) 108447, https://doi.org/10.1016/j.epsr.2022.108447.

[4] S. Cai, Z. Wang, F. Fuest, Y.J. Jeon, C. Gray, G.E. Karniadakis, Flow over an espresso cup: inferring 3-d velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks, J. Fluid Mech. 915 (2021) A102, https://doi.org/10.1017/jfm.2021.135.

[5] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks for heat transfer problems, J. Heat Transf. 143 (2021) 060801, https://doi.org/10.1115/1.4050542.

[6] F. Darlik, B. Peters, Reconstruct the biomass particles fields in the particle-fluid problem using continuum methods by applying the physics-informed neural network, Results Eng. 17 (2023) 100917, https://doi.org/10.1016/j.rineng.2023.100917.

[7] J. Guo, Y. Yao, H. Wang, T. Gu, Pre-training strategy for solving evolution equations based on physics-informed neural networks, J. Comput. Phys. 489 (2023) 112258, https://doi.org/10.1016/j.jcp.2023.112258.

[8] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems, Comput. Methods Appl. Mech. Eng. 365 (2020) 113028, https://doi.org/10.1016/j.cma.2020.113028.

[9] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier-Stokes flow nets): physics-informed neural networks for the incompressible Navier-Stokes equations, J. Comput. Phys. 426 (2021) 109951, https://doi.org/10.1016/j.jcp.2020.109951.

[10] G. Kissas, Y. Yang, E. Hwuang, W.R. Witschey, J.A. Detre, P. Perdikaris, Machine learning in cardiovascular flows modeling: predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 358 (2020) 112623, https://doi.org/10.1016/j.cma.2019.112623.

[11] S. Lee, B.T. Lee, S.K. Ko, Pinn application for compressible gas flow transient analysis, ICTC 2023 (2023) 1703–1708, https://ictc.org/sites/ictc2023a/media?key=site/ictc2023a/abs/P7-11.pdf.

[12] S. Lin, Y. Chen, The improved backward compatible physics-informed neural networks for reducing error accumulation and applications in data-driven higher-order rogue waves, arXiv preprint, arXiv:2312.06715, 2023.

[13] Q. Lou, X. Meng, G.E. Karniadakis, Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-bgk formulation, J. Comput. Phys. 447 (2021) 110676, https://doi.org/10.1016/j.jcp.2021.110676.

[14] D. Lucor, A. Agrawal, A. Sergent, Physics-aware deep neural networks for surrogate modeling of turbulent natural convection, ArXiv, arXiv:2103.03565 [abs], 2021.

[15] K.O. Lye, S. Mishra, D. Ray, Deep learning observables in computational fluid dynamics, J. Comput. Phys. 410 (2020) 109339, https://doi.org/10.1016/j.jcp.2020.109339.

[16] J. Magiera, D. Ray, J.S. Hesthaven, C. Rohde, Constraint-aware neural networks for Riemann problems, J. Comput. Phys. 409 (2020) 109345, https://doi.org/10.1016/j.jcp.2020.109345.

[17] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Eng. 360 (2020) 112789, https://doi.org/10.1016/j.cma.2019.112789.

[18] R. Mattey, S. Ghosh, A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations, Comput. Methods Appl. Mech. Eng. 390 (2022) 114474, https://doi.org/10.1016/j.cma.2021.114474.

[19] X. Meng, G.E. Karniadakis, A composite neural network that learns from multi-fidelity data: application to function approximation and inverse pde problems, J. Comput. Phys. 401 (2020) 109020, https://doi.org/10.1016/j.jcp.2019.109020.

[20] G.S. Misyris, A. Venzke, S. Chatzivasileiadis, Physics-informed neural networks for power systems, in: 2020 IEEE Power & Energy Society General Meeting (PESGM), 2020, pp. 1–5.

[21] S. Monaco, D. Apiletti, Training physics-informed neural networks: one learning to rule them all?, Results Eng. 18 (2023) 101023, https://doi.org/10.1016/j.rineng.2023.101023.

[22] C. Moya, G. Lin, Dae-pinn: a physics-informed neural network model for simulating differential algebraic equations with application to power networks, Neural Comput. Appl. 35 (2023) 3789–3804.

[23] V.M. Nguyen-Thanh, X. Zhuang, T. Rabczuk, A deep energy method for finite deformation hyperelasticity, Eur. J. Mech. A, Solids 80 (2020) 103874, https://doi.org/10.1016/j.euromechsol.2019.103874.

[24] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, https://doi.org/10.1016/j.jcp.2018.10.045.

[25] M. Raissi, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Deep learning of vortex-induced vibrations, J. Fluid Mech. 861 (2019) 119–137, https://doi.org/10.1017/jfm.2018.872.

[26] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations, Science 367 (2020) 1026–1030, https://doi.org/10.1126/science.aaw4741.

[27] L.F. Shampine, M.W. Reichelt, J.A. Kierzenka, Solving index-1 daes in Matlab and simulink, SIAM Rev. 41 (1999) 538–552, https://doi.org/10.1137/S003614459933425X.

[28] L. Sun, H. Gao, S. Pan, J.X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Eng. 361 (2020) 112732, https://doi.org/10.1016/j.cma.2019.112732.

[29] S.R. Vadyala, S.N. Betgeri, J.C. Matthews, E. Matthews, A review of physics-based machine learning in civil engineering, Results Eng. 13 (2022) 100316, https://doi.org/10.1016/j.rineng.2021.100316.

[30] H. Wessels, C. Weißenfels, P. Wriggers, The neural particle method – an updated Lagrangian physics informed neural network for computational fluid dynamics, Comput. Methods Appl. Mech. Eng. 368 (2020) 113127, https://doi.org/10.1016/j.cma.2020.113127.

[31] X.I.A. Yang, S. Zafar, J.X. Wang, H. Xiao, Predictive large-eddy-simulation wall modeling via physics-informed neural networks, Phys. Rev. Fluids 4 (2019) 034602, https://doi.org/10.1103/PhysRevFluids.4.034602.

[32] M. Yin, X. Zheng, J.D. Humphrey, G.E. Karniadakis, Non-invasive inference of thrombus material properties with physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 375 (2021) 113603, https://doi.org/10.1016/j.cma.2020.113603.