Marrying the benefits of Automatic and Numerical Differentiation in Physics-Informed Neural Network

Pao-Hsiung Chiu*
Institute of High Performance Computing
Singapore 138632
chiuph@ihpc.a-star.edu.sg

Jian Cheng Wong†
Institute of High Performance Computing
Singapore 138632
wongj@ihpc.a-star.edu.sg

Chin Chun Ooi
Institute of High Performance Computing
Singapore 138632
ooicc@ihpc.a-star.edu.sg

My Ha Dao
Institute of High Performance Computing
Singapore 138632
daomh@ihpc.a-star.edu.sg

Yew Soon Ong
School of Computer Science and Engineering
Nanyang Technological University
Singapore 639798
asysong@ntu.edu.sg

Abstract

In this study, a novel physics-informed neural network (PINN) is proposed to allow efficient training with improved accuracy. PINNs typically constrain their training loss function with differential equations to ensure outputs obey underlying physics. These differential operators are typically computed via automatic differentiation (AD), but this can fail with insufficient collocation points. Hence, the idea of coupling both AD and numerical differentiation (ND) is employed. The proposed coupled-automatic-numerical differentiation scheme (can-PINN) strongly links collocation points, thus enabling efficient training while being more accurate than simply using ND. As a demonstration, two instantiations of can-PINN were derived for the incompressible Navier-Stokes equations and applied to modeling of lid-driven flow in a cavity. Results show that can-PINNs can achieve very good accuracy even when the corresponding AD-based PINN fails.

1 Introduction

Physics-informed machine learning (1), in particular physics-informed neural networks (PINNs) have received increasing attention recently. The central idea of PINNs is to incorporate governing physical laws, typically differential equations, into the training loss function. PINNs have been demonstrated for various physics, including heat transfer (2; 3), fluid dynamics (4; 5; 6) and electromagnetics (7; 8; 9), and can be extended to inverse problems (10; 11; 12; 13). There have been significant efforts to improve PINN trainability (14; 15; 16; 17; 18; 19; 20; 21; 22). Nevertheless, training an accurate PINN remains a challenge (19; 20).

The vast majority of recent PINNs favor the fully connected architecture (23; 24; 25), where the computation of differential operators can be conveniently obtained via automatic differentiation (AD)

*Corresponding author, equal contribution
†equal contribution

Fourth Workshop on Machine Learning and the Physical Sciences (NeurIPS 2021).

(26). However, one can also compute these terms via numerical differentiation (ND), with very different impact on training. In the present study, we show that PINNs utilizing AD (a-PINNs) can only be accurately trained with sufficient collocation points. a-PINN training becomes delinked from solution accuracy under sparse sampling scenarios, i.e., even when the training losses are small, a-PINNs remain far away from the true solution. On the contrary, PINNs utilizing ND (n-PINNs) are robust and efficiently approximate the right solution with much less collocation points. However, they may be less accurate than a-PINNs given large quantities of collocation points in some cases.

This work introduces a novel coupled-automatic-numerical differentiation scheme for training PINNs (can-PINNs) that unifies the advantages of both a-PINNs and n-PINNs. The can-PINN formulation is generic and easily extended to various numerical schemes. To illustrate this, we present 2 versions based on the common upwind and central difference numerical schemes and provide intuition for their improved performance. We also demonstrate that can-PINNs robustly provide accurate solutions in sparse sample regimes where a-PINNs fail, while being more accurate than n-PINNs and matching the accuracy of a-PINNs in dense sample regimes when solving a lid driven flow in cavity. The sample codes can be found on https://github.com/chiuph/CAN-PINN.

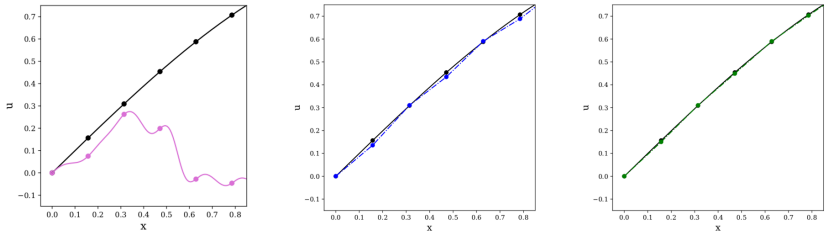
2 Methods

A detailed introduction to PINNs is provided in the Appendix for interested readers. Briefly, a typical PINN uses a fully connected architecture to model the dynamical system u(x,t;w) given spatial x $\in \Omega$ and temporal t $\in$ (0,T] inputs and network parameters w. Crucially, PINN training focuses on reducing the residual from the governing differential equations over a set of collocation points within the problem domain, thus ensuring that its output obeys underlying physics.

2.1 Issues with a-PINN and n-PINN training

While the PDE loss components are defined over a continuous domain, residuals are typically computed over a finite set of m collocation points $D = \{x_i, t_i\}_{i=1}^{m}$. When the PINN is higher order differentiable w.r.t. (x,t), differential operators in the PDE can be conveniently computed via AD which is already used during training, making AD the default for PINNs.

However, PINNs are commonly trained in an over-parameterized regime. As AD computes differential operators exactly at each individual collocation point and the AD-formulated loss function is under-constrained when the PINN is heavily over-parameterized, insufficient collocation points make a-PINNs susceptible to inaccurate solutions. Hence, the a-PINN may fulfill the underlying differential equation well at all collocation points, leading to a near zero training loss, even when its solution is different from the true solution (Figure 1a). Under such circumstances, the training loss is extremely misleading. This is particularly critical as PINN-type methods have been proposed for complex high-dimensional PDEs where dense sampling might be impractical (27).



(a) *a*-PINN (b) *n*-PINN (c) *can*-PINN

Figure 1: Schematics illustrating how (a) a-PINN almost perfectly matches the differential operator constraint at the collocation points (colored) but deviates from the true solution (black) while (b) n-PINN and (c) can-PINN approximate the true solution (black) by matching the gradient at piecewise local regions defined by support points surrounding the collocation points (colored).

One can employ ND in place of AD to alleviate this issue. The fundamental idea is to approximate derivatives by means of local support points via Taylor-series expansions. The resultant derivatives

are obtained by matching the values across local support points, thus enforcing a smooth transition across adjacent points. This allows n-PINNs to modulate gradient behavior at piece-wise local regions rather than isolated collocation points. As a result, n-PINNs learn the pattern in the entire solution space even with sparsely sampled collocation points (Figure 1b), although its accuracy depends on discretization and numerical scheme used.

2.2 can-PINN

Inspired by the multi-moment approach (28; 29; 30), we propose the coupled-automatic-numerical (can) differentiation method to augment the accuracy of n-PINNs. The idea is to approximate the derivative term $u_{x|can}$ by virtue of both u and u_x, where u_x is obtained from AD. To illustrate this, we derive the following based on the conventional 2^{nd} order upwind (uw2) and central difference (cd) schemes:

$$\frac{\partial u}{\partial x} \cong \frac{\partial u}{\partial x}|_{can(uw2)} = \frac{\hat{u}(x,t) - \hat{u}(x-\Delta x,t)}{\Delta x} + 1/2(\hat{u}_x(x,t) - \hat{u}_x(x-\Delta x,t)) \tag{1}$$

$$\frac{\partial p}{\partial x} \cong \frac{\partial p}{\partial x}|_{can(cd)} = \frac{\hat{p}(x+\Delta x,t) - \hat{p}(x-\Delta x,t)}{2\Delta x} - 1/8(\hat{p}_x(x+\Delta x,t) - 2\hat{p}_x(x,t) + \hat{p}_x(x-\Delta x,t)) \tag{2}$$

where the Δx is a parameter and may be defined as the distance between adjacent points. A modified equation analysis can be used to recast the above equations into the following forms:

$$\frac{\partial u}{\partial x}|_{can(uw2)} = \hat{u}_x - \frac{\Delta x^2}{12}\hat{u}_{xxx} + \frac{\Delta x^3}{24}\hat{u}_{xxxx} + ... \tag{3}$$

$$\frac{\partial p}{\partial x}|_{can(cd)} = \hat{p}_x + \frac{\Delta x^2}{24}\hat{p}_{xxx} - \frac{\Delta x^4}{480}\hat{p}_{xxxxx} + ... \tag{4}$$

From these equations, the proposed schemes can be seen as including additional stabilization terms to couple information from adjacent points. In addition, as the choice of local support points gets increasingly dense, Δx becomes vanishingly small, and the derivative is now equal to the derivative as computed by AD, ensuring the accuracy of the can-PINN formulation. As per typical numerical implementations, the upwind direction is defined based on the local flow field as obtained at each training iteration.

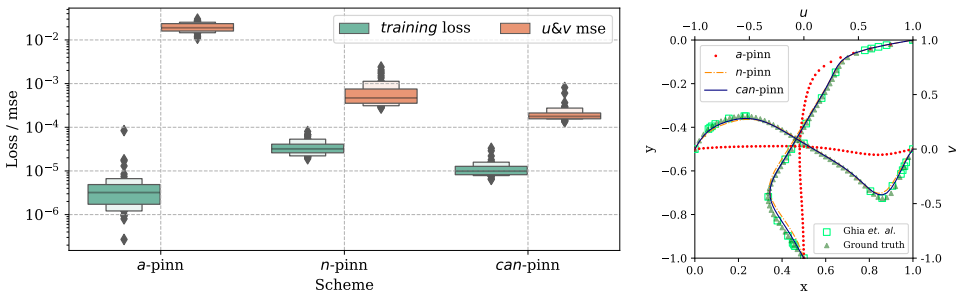
3 Results

3.1 Description of models for flow in a lid-driven cavity

The lid-driven cavity problem is a common benchmark for many numerical methods, due to its complex physics. This problem models flow within a square cavity with a lid velocity $u_{lid} = 1$ for the top wall, and zero velocity on other non-slip walls. The governing equations are the steady-state, 2-D incompressible N-S equations (in Appendix). To compute MSE, a reference solution is obtained by an in-house numerical solver based on the improved divergence-free condition compensated (IDFC) method (31). Additional information on the PINN architecture used is in the Appendix.

3.2 Performance of a-PINN, n-PINN and can-PINN

The performance of a-PINN, n-PINN, and can-PINN for 50 independent runs when trained with 2601 collocation points sampled from a 51x51 grid are compared in terms of training loss and solution MSE (Figure 2a). Despite the lowest training loss, the a-PINN's solutions are consistently bad. Their MSEs (>1e-2) are more than 1 order of magnitude higher than those obtained by n-PINN, and are about 2 orders of magnitude higher than can-PINNs. The results also show that the proposed can-PINNs are significantly more accurate than n-PINN. In Figure 2b, we plot the velocity profiles along the cutting lines at the center of the cavity. Good agreement is obtained for the can-PINN with both our in-house simulation and a literature benchmark (32). The n-PINN profiles deviate slightly more from the simulation and benchmark, while a-PINN results show a large discrepancy. Additional contour plots illustrating the relative performance of the PINN models are in the Appendix.

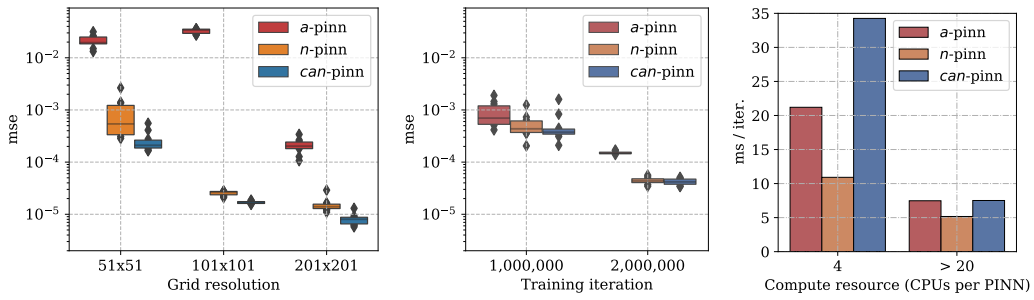


(a) Distribution of training loss and u&v MSE (b) Velocity profile along center cutting lines

Figure 2: (a) Distribution of training loss and velocity MSE across different PINNs from 50 independent runs. (b) The velocity profiles along the center cutting lines, from the median solution obtained are compared to the simulation and benchmark results from (32).

3.3 Training under different sampling scenarios

As the previous results suggest that sampling collocation points from a 51x51 grid is insufficient for training a good a-PINN model, we further study the performance of different PINNs with collocation points from a denser selection of points, i.e., 101x101 and 201x201 points. The distribution of solution MSE based on 10 independent runs for different PINNs and sampling scenarios are presented in Figure 3(a). The results show that all 3 PINN models achieve a more accurate solution when the collocation points are sampled from a denser selection. However, the a-PINN only achieves a good solution with MSE ($\sim$2e-4) on the finest resolution of 201x201. Even then, this result is only on par with the can-PINN on a 51x51 grid, and out-performed by can-PINN on a 201x201 grid. In addition, this requires both double the mini-batch size and much more training iterations than the can-PINN. Also, to ensure a fair comparison, the Δx parameter employed for can-pinn was chosen to be identical to the spacing used in generating the set of collocation points for each experiment, ensuring that the introduction of the can-PINN ansatz does not provide additional training points that the a-PINN does not see.



(a) Fix grid sampling (b) Uniform distribution sampling (c) Training time per iteration

Figure 3: (a) Distribution of velocity MSE between PINNs for collocation points sampled from 51x51, 101x101, and 201x201 grids. (b) Distribution of velocity MSE vs. training iteration, with collocation points randomly sampled from uniform distribution. (c) Comparison of PINNs' training time (averaged from 5,000 iterations) for 500 mini-batch samples, under limited and excessive compute resource scenarios.

The performance of different PINNs when trained on collocation points randomly sampled from a uniform distribution is also studied. Since the collocation points are not uniformly spaced, the Δx parameter used in the can-PINN loss function is less well-defined. Δx=0.01 is used for this work. In the limit as training iteration grows, this random uniform sampling is equivalent to an infinitely dense set of collocation points. Figure 3(b) compares the MSE at 1 and 2 million training iterations. Again, all 3 PINNs achieve a more accurate solution after more training iterations. a-PINN performs

better on the uniform distribution sampling given the same large amount of training iteration. On the contrary, n-PINN and can-PINN training is consistently more efficient under the fixed grid sampling scenario. Critically, can-PINN consistently outperforms a-PINN and n-PINN across all scenarios. The above results indicate that PINNs generally achieve a better solution with increased sampling resolution, albeit with more training iterations. Importantly, this increase in training iterations can be a practical bottle-neck, especially when optimizing hyper-parameters.

Finally, we report the per iteration training time from different PINN models as used in the present study as per Figure 3(c) for reference. The PINN implementations used the Keras API as packaged with TensorFlow2.5. During training, n-PINN requires only forward pass for the computation of the differential operators for loss evaluation, which can be faster than the back-propagation AD computation utilized by a-PINN. The can-PINN however performs both forward pass and back-propagation during the loss evaluation. Hence, with a limited compute resource (i.e., 4 CPUs per PINN), the n-PINN is the quickest, while the can-PINN is the slowest. In addition, Keras and its backend TensorFlow automatically parallelize execution. Hence, when there are over 20 CPUs per PINN, the a-PINN, n-PINN, and can-PINN show similar execution times per iteration. Critically, we note that the n-PINNs and can-PINNs are also generally more sample efficient and converge with less total iterations than the a-PINNs, in addition to having similar execution times per iteration.

4 Conclusion

In this work, we studied the difference between PINNs with training loss computed by AD and ND. It was observed that the AD-formulated loss function is an under-constrained optimization problem, which causes a-PINN training to fail without adequate sampling, while the n-PINN is much more efficient and robust to sampling resolution.

Inspired by the multi-moment approach, we further proposed a coupled-automatic-numerical differentiation method that utilizes both AD and ND principles. The resultant can-PINN is much more sample efficient and yields improved accuracy. The proposed can-PINN is a generic framework and can be extended to many coupled-automatic-numerical schemes of varying form and accuracy. With the application to fluid dynamic problems in mind, we derived two can-PINN schemes based on the upwind and central difference numerical schemes, and demonstrate the ability for can-PINN to efficiently train on sparse samples while robustly producing an accurate solution. This new method can potentially enable the extension of PINNs to even more complex problems where conventional a-PINN formulations might be challenging to train.

Acknowledgments and Disclosure of Funding

This research is supported by A*STAR under its AME Programmatic programme: Explainable Physics-based AI for Engineering Modelling and Design (ePAI) Award No. A20H5b0142.

References

[1] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, *Nat. Rev. Phys.* 2021 36. 3 (2021) 422–440.

[2] N. Zobeiry, K.D. Humfeld, A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications, *Eng. Appl. Artif. Intell.* 101 (2021) 104232.

[3] S. Amini Niaki, E. Haghighat, T. Campbell, A. Poursartip, R. Vaziri, Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture, *Comput. Methods Appl. Mech. Eng.* 384 (2021).

[4] 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.

[5] 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).

[6] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, *Comput. Methods Appl. Mech. Eng.* 360 (2020).

[7] Z. Fang, J. Zhan, Deep Physical Informed Neural Networks for Metamaterial Design, *IEEE Access.* 8 (2020) 24506–24513.

[8] P. Zhang, Y. Hu, Y. Jin, S. Deng, X. Wu, J. Chen, A maxwell's equations based deep learning method for time domain electromagnetic simulations, *Proc. 2020 IEEE Texas Symp. Wirel. Microw. Circuits Syst. Mak. Waves Texas, WMCS 2020.* (2020).

[9] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural network methods in quantum mechanics, *Comput. Phys. Commun.* 104 (1997) 1–14.

[10] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, *Science.* 367 (2020) 1026–1030.

[11] 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).

[12] M. Raissi, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Deep learning of vortex-induced vibrations, *J. Fluid Mech.* 861 (2019) 119–137.

[13] K. Shukla, P.C. Di Leoni, J. Blackshire, D. Sparkman, G.E. Karniadakis, Physics-Informed Neural Network for Ultrasound Nondestructive Quantification of Surface Breaking Cracks, *J. Nondestruct. Eval.* 2020 393. 39 (2020) 1–20.

[14] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks, *Comput. Methods Appl. Mech. Eng.* 384 (2021) 113938.

[15] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, G. Wetzstein, Implicit Neural Representations with Periodic Activation Functions, *Adv. Neural Inf. Process. Syst.* 33 (2020) 7462–7473.

[16] R. van der Meer, C. Oosterlee, A. Borovykh, Optimally weighted loss functions for solving PDEs with Neural Networks, (2020). https://arxiv.org/abs/2002.06269v4

[17] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, (2020). https://arxiv.org/abs/2001.04536v1

[18] S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, (2020). https://arxiv.org/abs/2007.14527v1

[19] L. McClenny, U. Braga-Neto, Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism, (2020). https://arxiv.org/abs/2009.04544v2

[20] M.A. Nabian, R.J. Gladstone, H. Meidani, Efficient training of physics-informed neural networks via importance sampling, *Comput. Civ. Infrastruct. Eng.* 36 (2021) 962–977.

[21] J.C. Wong, A. Gupta, Y.S. Ong, Can Transfer Neuroevolution Tractably Solve Your Differential Equations?, IEEE Comput. Intell. Mag. 16 (2021) 14–30. https://doi.org/10.1109/MCI.2021.3061854.

[22] J.C. Wong, C. Ooi, A. Gupta, Y.-S. Ong, Learning in Sinusoidal Spaces with Physics-Informed Neural Networks, (2021). https://arxiv.org/abs/2109.09338v1 (accessed September 21, 2021).

[23] O. Hennigh, S. Narasimhan, M.A. Nabian, A. Subramaniam, K. Tangsali, M. Rietmann, J. del A. Ferrandis, W. Byeon, Z. Fang, S. Choudhry, NVIDIA $SimNet^{TM}$: an AI-accelerated multi-physics simulation framework, (2020) 447–461. https://arxiv.org/abs/2012.07938v1

[24] E. Haghighat, R. Juanes, SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks, *Comput. Methods Appl. Mech. Eng.* 373 (2021) 113552.

[25] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: A Deep Learning Library for Solving Differential Equations, *SIAM Rev.* 63 (2021) 208–228.

[26] A. Güne¸, G. Baydin, B.A. Pearlmutter, J.M. Siskind, Automatic Differentiation in Machine Learning: a Survey, *J. Mach. Learn. Res.* 18 (2018) 1–43.

[27] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, *J. Comput. Phys.* 375 (2018) 1339–1364.

[28] F. Xiao, Unified formulation for compressible and incompressible flows by using multi-integrated moments I, *J. Comput. Phys.* 195 (2004) 629–654.

[29] K. Yokoi, M. Furuichi, M. Sakai, An efficient multi-dimensional implementation of VSIAM3 and its applications to free surface flows, *Phys. Fluids.* 29 (2017) 121611.

[30] P.H. Chiu, H.J. Poh, Development of an improved divergence-free-condition compensated coupled framework to solve flow problems with time-varying geometries, *Int. J. Numer. Methods Fluids.* 93 (2021) 44–70.

[31] P.H. Chiu, An improved divergence-free-condition compensated method for solving incompressible flows on collocated grids, *Comput. Fluids.* 162 (2018) 39–54.

[32] U. Ghia, K.N. Ghia, C.T. Shin, High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method, *J. Comput. Phys.* 48 (1982) 387–411.

A Appendix

A.1 Physics-Informed Neural Networks

In this section, we briefly outline the PINN methodology as commonly employed. A typical PINN uses a fully connected architecture to predict the dynamical process u(x,t;w) given spatial x $\in \Omega$ and temporal t $\in$ (0,T] inputs. The PINN's accuracy is determined by the network parameters w, which are optimized w.r.t. during training. The system u is assumed to be described by differential equations of the general form:

$$N_t[u(x,t)] + N_x[u(x,t)] = 0 \qquad x \in \Omega, t \in (0,T] \tag{5a}$$

$$u(x,0) = u_o(x) \qquad x \in \Omega \tag{5b}$$

$$B[u(x,t)] = g(x,t) \qquad x \in \partial\Omega, t \in (0,T] \tag{5c}$$

where N_t and N_x are general differential operators which can include any combination of linear and non-linear temporal and spatial derivatives. The IC at t=0 is defined by u_o (x). The boundary operator, B, can be any Dirichlet, Neumann or Robin BC, and enforces the desired condition g(x,t) at the boundary $\partial\Omega$. The PINN training loss function is then defined as:

$$L = \lambda_{DE} L_{DE} + \lambda_{IC} L_{IC} + \lambda_{BC} L_{BC} \tag{6a}$$

$$L_{DE} = \left\| N_t[u(x,t;w)] + N_x[u(x,t;w)] \right\|^2_{\Omega \times (0,T]} \tag{6b}$$

$$L_{IC} = \left\| u(x,0;w) - u_0 \right\|^2_{\Omega} \tag{6c}$$

$$L_{BC} = \left\| B[u(x,t;w)] - g(x,t) \right\|^2_{\partial\Omega \times (0,T]} \tag{6d}$$

which includes the PDE loss component, L_{DE}, and the IC and BC loss components, L_{IC} and L_{BC}. The relative weights, λs, control the trade-off between different components in the loss function. To further show the reliability and efficiency, the data loss component is not included in loss function, and all λs are set to 1 in this study.

A.2 Navier-Stokes Equations

The governing equations used in our PINN formulation are the steady-state, 2-D incompressible N-S equations:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{7a}$$

$$\frac{\partial (uu)}{\partial x} + \frac{\partial (vu)}{\partial y} = \frac{1}{Re}\left(\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y}\right)\right) - \frac{\partial p}{\partial x} \tag{7b}$$

$$\frac{\partial (uv)}{\partial x} + \frac{\partial (vv)}{\partial y} = \frac{1}{Re}\left(\frac{\partial}{\partial x}\left(\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial v}{\partial y}\right)\right) - \frac{\partial p}{\partial y} \tag{7c}$$

In the above equations, the primitive variables (u,v) and p are velocity and pressure and Re is Reynolds number. We solve for the lid driven cavity problem at Re = 400 within a unit-length spatial domain.

A.3 PINN architecture and training details

All experiments are run with the standard Tensorflow and Keras implementation on a 20-core workstation with an Intel Xeon Gold 6248 processor and an Nvidia RTX 2080 Ti GPU. However, our initial experiments revealed that the GPU was not significantly faster than the CPU, and all reported experiments in this work were run on the CPU.

As per in Figure 4, the general PINN architecture and training setting used is as follows. The network is comprised of 7 hidden layers, with the two spatial variables (x,y) expanding to 4 hidden layers with 64, 20, 20, 20 nodes in each layer, before branching out to another set of 3 hidden layers with 20 nodes in each layer, that predict each of (u,v,p). We incorporate the sinusoidal mapping:

$$\gamma(\nu) = \sin(2\pi(\boldsymbol{W}\nu + \boldsymbol{b})) \tag{8}$$

that act on PINNs' 2-dimensional spatial inputs $\nu = [x,y]^T$, into the first hidden layer of PINN, and initialize its weights $\boldsymbol{W}$ by sampling from a normal distribution $N(0,\sigma^2), \sigma = 1$. The bias

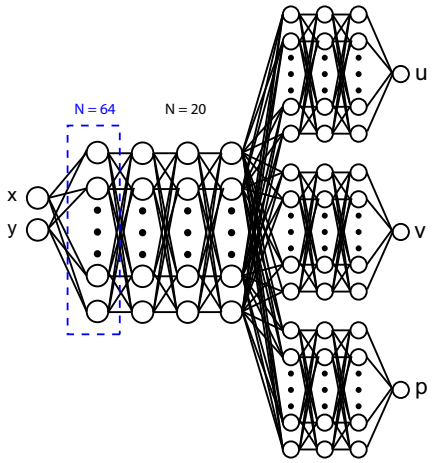
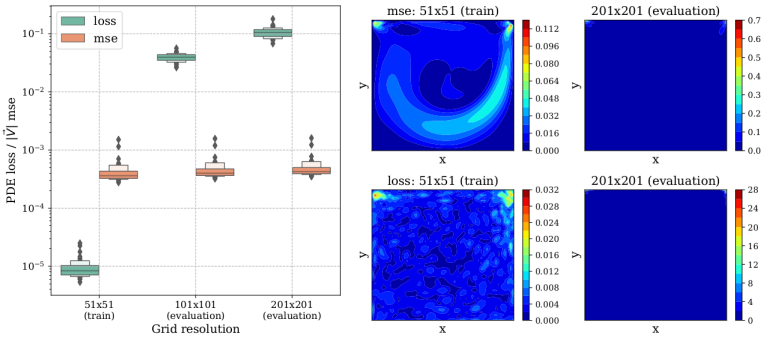


Figure 4: Schematic of the PINN architecture for the present study.

$\boldsymbol{b}$ is initialized as a zero vector. The subsequent hidden layers also use "sine" activation, except for a "linear" activation function in the output layer. Their weights are initialized by He uniform distribution. We reduce the learning rate on plateauing starting from an initial learning rate of 1e-3 until a min. learning rate of 5e-6 is reached. Each test problem was trained for 200,000 total training iterations, and the performance of a-PINNs, n-PINNs, and can-PINNs was compared under identical network architectures and training settings. Different activation functions were also tested in this work. With the commonly used activation function tanh, a-PINNs optimization is still unsuccessful while can-PINNs and n-PINNs are successful, while can-PINNs still achieves better error MSE than n-PINNs.

In addition, it is important to consider the physical nature when approximating the different derivative terms, hence, convective terms are frequently better approximated by upwind schemes, while pressure gradients can be well approximated by central difference schemes. Consequently, the upwind-based can(uw2) and can(cd) scheme, and second order upwind scheme and central difference scheme are employed for convection and pressure gradient terms for the can-PINNs and n-PINNs respectively. All other differential terms are approximated by a central difference scheme.

For the sake of completeness, we also plotted the residuum plot for the training grid and higher resolution grid. From Figure 5, it reveals that high residual appears near the boundary where the physics is more complex. Nevertheless, the solution MSE is stable even the residual MSE increases by a few orders of magnitude.

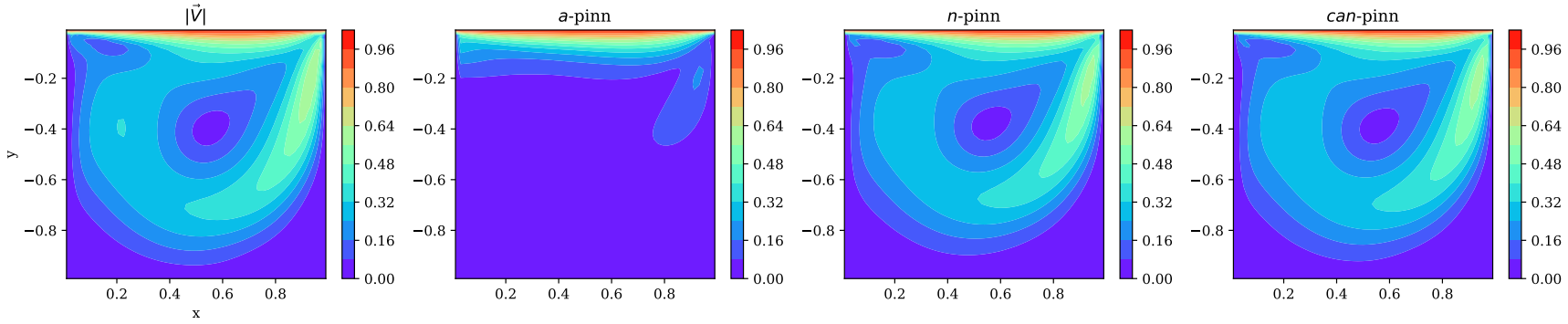


(a) (b)

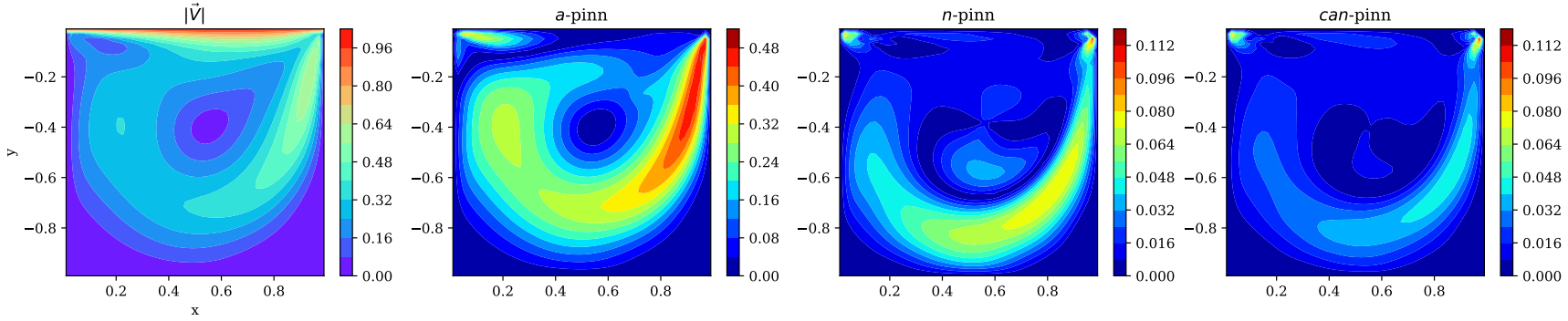
Figure 5: (a) Distribution of residuum for trained grid and higher resolution grids. (b) Contour plot of residuum for trained grid and higher resolution grids.

A.4 Velocity contour of results

Figure 6 compares the velocity magnitude contours computed by a-PINN, n-PINN, and can-PINN, and their deviation from the solution. Clearly, it is difficult to obtain a reasonable solution by a-PINN with the current 51x51 grid, i.e., the correct flow does not develop. While both the n-PINN and can-PINN resemble the ground truth, the can-PINN is clearly more accurate.



(a) Ground truth velocity magnitude $|\vec{V}|$ vs. PINN solutions



(b) Ground truth velocity magnitude$|\vec{V}|$ and absolute error of PINN solutions

Figure 6: (a) Comparison between the ground truth velocity magnitude and the solutions solved by a-PINN, n-PINN and can-PINN. (b) The absolute error between the PINN solutions and ground truth for the solution with median MSE from 50 independent runs.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Instructions required to reproduce the results are described in the text. Sample codes can be found on https://github.com/chiuph/CAN-PINN
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Training details are described in the Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Graphs are plotted with error bars for repeated runs
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Compute used are described in the Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]