Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

On Adaptive Loss Weights of Physics Informed Neural Network

Afrah Farea[1] | M.Serdar Çelebi[1] | Emre Ersan[1] | Reza Daryani[1]

[1] Email: farea16,mscelebi,ersane,daryani@itu.e
Istanbul Technical University, Department of Computational Science and Engineering, Istanbul, Turkey

Correspondence
*Afrah Farea, Computational Science and Engineering Department, Informatics Institute, Istanbul, 34469, Turkey. Email: farea16@itu.edu.tr

Summary

This paper explores the ability of the Physics Informed Neural Network to solve nonlinear partial differential equations. Despite the success of this method, the vanilla PINN algorithm may produce inaccurate results or even fail to converge entirely, especially with large-scale systems. This is mainly attributed to using the gradient descent method to minimize a multi-objective loss function with multiple competing terms. The complexity increases with multiple outputs of the neural network. In this sense, adaptive methods are necessary to balance the different terms in the loss function and enforce the neural network to fit the solution of the PDEs. Experiments show that different outputs of the neural network should also have different weights. We investigate the different adaptive methods to enhance the performance of PINN on selected flow field problems. We examine the effect of adaptive weighting methods on the different outputs of the neural network and the different loss terms.

KEYWORDS:

1 | INTRODUCTION

Computational fluid dynamics (CFD) are among the most crucial modelling methods of fluid simulation and have many interesting applications in various industry and engineering disciplines such as computer graphics, liquid simulations and medical applications. Such systems are generally nonlinear and require complex scientific simulations that notoriously generate large amounts of data. A set of differential equations, such as Navier-Stokes equations, usually describes these simulations. Like many other differential equations, large classes of these equations lack analytical solutions, so numerical approximations are usually used, such as finite methods (FM), spectral and spline methods. These techniques require the computational domain to be discretized into a set of mesh points, and the solution is only approximated by solving Navier-Stokes equations at these points. This introduces truncation errors, especially with complex geometries. The cost of these solutions increases exponentially with the increasing dimension of the problem in what is known as the curse of dimensionality.

Like many scientific areas, partial differential equations have seen a massive surge in experimental and computational studies using machine learning techniques. The result is a new field of science where machine learning tools are used to solve complex scientific problems. As a new field of study, different terminologies are used for this new field such as scientific machine learning (SciML), Deep Learning for Science, Physics-informed deep learning, Physics-informed machine learning.

Physics-informed neural networks (PINN) have recently been an active research topic for the scientific machine learning community thanks to their flexibility in tackling forward and inverse problems efficiently within the same optimization problem to learn a single solution of a function. To learn a family of solutions (also called function operator), multiple simulations are required, and the network is trained by averaging on these simulation samples. Examples of such operators include DeepOnet[1], Fourier neural operator[2], and Physics informed neural operator[3].

The contribution of this paper can be summarized as follows:

-

The rest of the paper is organized as follows: section 2 introduces PINN and adaptive methods used in the study, experimental analysis and results are shown in section **??**. Finally, we conclude the present work in section 3.

2 | PHYSICS INFORMED NEURAL NETWORKS (PINN)

The term physics-informed neural networks was first coined in[4] to refer to training neural networks for solving general differential equations of the form,

$$\begin{aligned}\mathcal{D}[u(t,x);\alpha] &= f(t,x), \quad t \in [t_0, T], \quad x \in \Omega, \\ \mathcal{B}_k[u(t,x)] &= g_k(t,x), \quad t \in [0,T], \quad x \in \Gamma_k \subset \partial\Omega, \quad k = 1,2,\dots,n_b\end{aligned} \tag{1}$$

Where $\mathcal{D}$ is the differential operator, $\mathcal{B}_k$ is a set of boundary operators, $f(x)$ is the forcing function, $g_k(x)$ is a set of boundary functions, α is a set of parameters used in the differential equation and $u(x)$ is the solution of the differential equation given input x in the solution domain Ω with boundary $\partial\Omega$.

Figure 1 shows a general schematic of the PINN model. The neural network output is used explicitly to get the derivatives of the approximate solution concerning the input variables by applying the chain rule using automatic differentiation. The idea is to recover the unique solution of the PDE by penalizing the boundary and initial conditions as supervised learning and selecting finitely many representative points from the solution domain as regularization terms to satisfy the PDE equation.Throughout this work, we refer to the PDE loss as the physics loss.

While the physical loss ensures that the neural network solution captures knowledge about the governing PDE equation by minimizing the residual of the differential equation on a set of randomly selected representative points, the boundary loss ensures that the neural network solutions satisfy the proper unique solution of the differential equation on the selected boundary points. Hence, the problem is a multi-objective objective optimization problem (MOOP)[?]. The most direct technique to solve this problem is to aggregate them into a single loss function:

$$\mathcal{L}(\theta) := argmin_{\theta} \sum_{k=0}^{n} \mathcal{L}_k(\theta) = argmin_{\theta}\Big(\mathcal{L}(\mathcal{D}[u(t,x);\alpha] - f(t,x)) + \sum_{k=1}^{n_b} \mathcal{L}(\mathcal{B}[u(t,x)] - g_k(t,x))\Big) \tag{2}$$

where n is the number of terms in the loss function. In fact, this is a soft way of imposing constraints because the learned model may violate some of these constraints since the objective is to minimize the loss function as a whole. A key observation in this equation is that it consists of terms with different physical scales. This is because the physical loss may contain high-order derivatives, mainly the physical loss terms, that can impact the quality of training[?]. The straightforward approach to balancing different parts of the loss function is to multiply each term by a weighting coefficient λ_k during the training process. Therefore, the loss function can be:

$$\mathcal{L}(\theta) := argmin_{\theta} \sum_{k=0}^{n} \lambda_k \mathcal{L}_k(\theta) = argmin_{\theta}\Big(\lambda_0 \mathcal{L}(\mathcal{D}[u(t,x);\alpha] - f(t,x)) + \sum_{k=1}^{n_b} \lambda_k \mathcal{L}(\mathcal{B}[u(t,x)] - g_k(t,x))\Big) \tag{3}$$

where λ_k is the weighting coefficient of the loss term.

This method is used in the baseline PINN algorithm. However, it turns out that selecting the weight values is essential to avoid failure and accelerate convergence, especially with highly nonlinear and chaotic systems. As an example, Table 1 shows the effect of assigning different weight values to the boundary and the physical loss terms on the solution of the 1D-Poisson equation. However, manual assignment can make the solution quality dependent on tedious hyperparameter tuning.

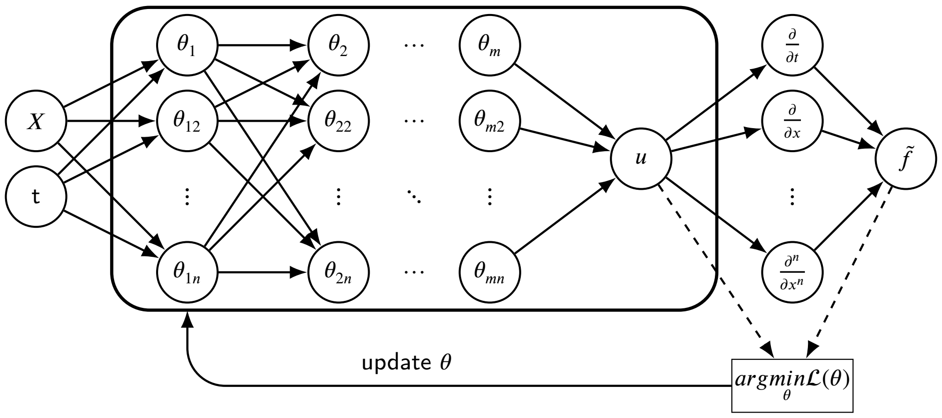


FIGURE 1 Schematic of PINN model

λ_{phy}	λ_{bc}	MSE_u	MSE_r	time(S)
0	1	1.07	0.976	74.50
1	1	0.058	0.0067	54.81
1	2	0.025	0.0056	53.51
1	10	0.0042	0.0053	53.43
1	50	0.0015	0.007	52.06
1	100	0.00091	0.0058	52.87
1	**500**	**0.0004**	**0.006**	**60.35**
1	1000	_	_	NaN
1	0	4.44	0.0050	60.33
2	1	0.0384	0.0031	59.86
2	10	0.0022	0.00265	50.28
10	1	0.0195	0.0099	63.118
10	50	0.013	0.0023	58.034
50	1	_	_	NaN
50	50	_	_	NaN

TABLE 1 Effect of different weight values on the PINN for the solution of 1D-Poisson equation using 128 mini-batch with TensorFlow Gradient Descent Optimizer and one hidden layer with 500 neurons.

Many extensions of PINN have been proposed to enhance the baseline algorithm. Below, we summarize some notable works on adaptive loss weight methods.

2.0.1 | Using temporal dataset information:

Instead of using the entire space-time at once for training. Krishnapriyan et al.[5] suggest dividing the training space into time slices Δt and training the model in a sequence-to-sequence manner. The motivation is that learning the whole solution domain can be difficult, especially when the simulation is time-driven. The initial condition is handled by feeding the result from the previous time slice, while the given initial condition is fed at the first time slice. The problem with this approach is that the gradient may decrease rapidly with increasing number of iterations. Therefore, if we train the simulation dataset sequentially in a time-marching manner, the gradient will be significant at the beginning of the simulation (the first time slice) and then fade quickly. Hence, training may not be practical for later timesteps. This is significantly common when solving transient problems[6]. A similar approach is suggested in[7]. Rather than using a single neural network to train the time sequences, they also suggest using separate neural networks for each time slice and the initial condition from the precision of the previous time slice.

Wang et al.[8] proposed an annealing weighting strategy by adding an average of adaptive weights to the PDE physical loss term. The weights are a function of the loss term at different time steps such that,

Problem	n	$\overline{\lambda}$	$RMSE$	time(S)
1D Wave	10	$\overline{\lambda_{bc}}$: 6.4083e+08	u: 1.72e+00	6.87e+03
		$\overline{\lambda_{ic}}$: 2.5778e+12		
	100	$\overline{\lambda_{bc}}$: 2.3740e+08	u: 0.936	6555.516
		$\overline{\lambda_{ic}}$: 7.0695e+08		
	1000	$\overline{\boldsymbol{\lambda}_{\mathbf{bc}}}$ **: 7.3042e + 04**	**u : 0.0083**	**6490.177**
		$\overline{\boldsymbol{\lambda}_{\mathbf{ic}}}$ **: 2.8042e + 03**		
	5000	$\overline{\lambda_{bc}}$: 1.1316e+04	u: 0.0348	6469.520
		$\overline{\lambda_{ic}}$: 3.8878e+02		
Helmholtz	10	$\overline{\lambda_{bc}}$: 6.8432e+01	u: 2.4850e-02	3.2207e+02
			f: 1.1853e-02	
	100	$\overline{\lambda_{bc}}$: 8.6759e+01	u: 1.7900e-02	5.7906e+02
			f: 1.3365e-02	
	1000	$\overline{\lambda_{bc}}$: 6.7495e+01	u: 1.8577e-02	6.5756e+02
			f: 1.4351e-02	
	5000	$\overline{\boldsymbol{\lambda}_{\mathbf{bc}}}$ **: 5.6556e + 01**	**u : 1.2671e − 02**	**6.5169e + 02**
			f : 1.1068e − 02	
Klein Gordon	10	$\overline{\lambda_{bc}}$: 2.1222e+02	u: 0.01150	461.579
		$\overline{\lambda_{ic}}$: 4.5766e+01	f: 0.00306	
	100	$\overline{\lambda_{bc}}$: 2.4772e+02	u: 0.0065	434.199
		$\overline{\lambda_{ic}}$: 4.6157e+01	f: 0.0026	
	1000	$\overline{\boldsymbol{\lambda}_{\mathbf{bc}}}$ **: 2.0478e + 02**	**u : 8.5268e − 03**	**553.211**
		$\overline{\boldsymbol{\lambda}_{\mathbf{ic}}}$ **: 6.4414e + 01**	**f : 8065e − 03**	
	5000	$\overline{\lambda_{bc}}$: 1.6037e+02	u: 0.0197	453.269
		$\overline{\lambda_{ic}}$: 5.1508e+01	f: 0.0038	
Lid-driven Cavity	10	$\overline{\lambda_{bc}}$: 1.1164e+01	u: 1.7975e-01	7.6789e+03
			v: 2.9160e-01	
	100	$\overline{\lambda_{bc}}$: 1.0727e+01	u: 1.5258e-01	3.3917e+03
			v: 2.7332e-01	
	1000	$\overline{\boldsymbol{\lambda}_{\mathbf{bc}}}$:1.0746e+01	u: 2.0277e-01	3.6462e+03
			v: 3.1740e-01	
	5000	$\overline{\boldsymbol{\lambda}_{\mathbf{bc}}}$ **: 5.2003e + 00**	**u : 1.1205e − 01**	**3.5537e + 03**
			v : 2.1191e − 01	

TABLE 2 Effect of updating weights every n iterations on the performance of PINN using model $M2$ of[9] on some selected problems where $\overline{\lambda}$ is the average calculated value of λ, $RMSE$ is the relative mean square error of the test cases. $\overline{\lambda_{phy}}$ is set to one.

$$\mathcal{L}_r^k = \frac{1}{N_t} \sum_{i=0}^{N_t} \lambda_i \mathcal{L}_r^k(t_i, \theta) \tag{4}$$

where N_t is the number of training instances, $\lambda_i = exp\Big(-\epsilon \sum_{k=1}^{i-1} \mathcal{L}_r^{k-1}(t_k, \theta) \Big)$ where k is the iteration number.

This method requires a large memory and computational cost, especially with gradient descent approaches, because it assigns weights to each training instance. It is also sensitive to the initial value of the hyperparameter ϵ.

Using gradient information:

Wang et al.[9] attributed the failure of the PINN model to the stiffness in the gradient flow dynamics where the gradient of the terms, mostly the boundary terms, approaches zero during training. Inspired by gradient descent formula, they developed an adaptive annealing mechanism that utilizes the back-propagation gradient statistics during model training to balance the different terms of the loss function:

$$\lambda_i^k = (1-\alpha)\lambda_i^{k-1} + \alpha \frac{max_\theta\{|\nabla_\theta L_r(\theta_n)|\}}{mean\{|\nabla_\theta L_i(\theta_n)|\}} \qquad i = 1, 2, ..., n_b \tag{5}$$

bar o where n_b is the number of the loss terms, k is the iteration number, α is the hyperparameter value that describes the amount of contribution of the current and previous value of λ_i. In a separate experiment, we reproduce results in[9] using their code where the λ values are updated at different iteration numbers. The results are shown in table 2. Except for the 1D-wave case, the algorithm is not very sensitive to the frequency updates of the weight values. One can enhance the algorithm by updating the term weights only when the gradient of the term is lower than a certain threshold value. This helps save computation time for calculating the gradient. Experiments also show that the gradient still approaches zero, especially in the hidden layers. According to[?], it is recommended to replace the mean value with other metrics such as Kurtosis and standard deviation.

Recent studies suggest that the fully connected neural network used in PINN makes it difficult to learn functions with high frequencies due to the "spectral bias" effect of the neural network[10 8, 11]. At the beginning of training, the eigenvalues of the NTK matrix are large, corresponding to larger wavelengths and, hence, smaller frequency values[8]. Large eigenvalues also correspond to a faster convergence rate. However, these eigenvalues are getting smaller with increasing number of training iterations. Therefore, neural networks learn high frequencies first and capture the lower frequencies with increasing iterations. In their subsequent work, Wang et al.[6] developed an adaptive strategy that makes use of the eigenvalues of a derived NTK of PINNs and proved that under certain conditions, the NTK of PINNs converges to a deterministic kernel and stays constant during training in the infinite width limit for any neural network architecture and any PDE operator. The weights are updated according to the following formula,

$$\lambda_i^k = \frac{\sum_j^{n_b} Tr(K_j^k)}{Tr(K_i^k)} \qquad i = 1, 2, ..., n_b \tag{6}$$

Where $Tr(.)$ is the trace, k is the iteration number, and K is the NTK matrix of the network parameters with respect to the corresponding term. The NTK matrix is empirically set to the Jacobean matrix of the term times its transpose. We refer to this model as the M_3 model.

Using training dataset information:

Wight et al.[7] propose an adaptive sampling approach that increases the number of training instances in regions with high prediction errors and suggest an adaptive sampling method by adding more training instances from the regions where the prediction error is high. McClenny et al.[12] suggest an adaptive weighting mechanism by adding adaptive weights to each training instance and updating them along with the network weights. The loss functions minimizied with respect to the network weights and maximized with respect to the adaptive weights. This method is Ineffective with large training datasets, especially when stochastic gradient descent. Instead of assigning weights to each training instance, Ref[13] assign weights to each loss function and use the same updating method to update these weights.

Other methods:

Japtap et al.[14] proposed using adaptive activation functions for training. Ji et al.[15] studied the performance of PINN on selected stiff ODE problems generated from chemical kinetic systems. They argue that the performance of PINN was enhanced by removing the stiff terms from the PINN loss function. Costabal et al.[16] suggest an adaptive data sampling method based on maximum entropy measurement. To mitigate the spectral bias problem of the neural networks as well as enhance PINN performance and convergence rate on large-scale problem sizes, Moseley et al.[17] proposes finite basis PINN (FBPINN) inspired by classical finite element methods where the solution domain is divided into a finite set of overlapping subdomains and learn each subdomain with PINN. The motivation is that high-frequency features in large domains become low-frequency features in smaller domains and thus reduce the spectral bias effect of the neural network.

Table 3 summarizes the model description used in the study.

Model	Description
M_1	fixed weighting
M_2	adaptive weighting using gradient[9]
M_3	adaptive weighting using NTK[6]
M_4	adaptive weighting using training error of the previous step

TABLE 3 Models considered in the study

TABLE 4

	Method		Model				
Problem	Gradient Descent	Metric	$M1$[?]	$M2$[?]	$M3$[?]	$M4$[?]	$M5$
1D-Wave[?]	mini-batch	RL_2 u	0.0428	0.005	0.0041	0.0475	
		Time	4326.068	2242.01	3124.91	1542.90	
	full-batch	RL_2 u				0.0294	
		Time				14959.167	
Helmholtz[?]	mini-batch	RL_2 u	0.23	0.03		0.117	0.055
		RL_2 f	0.0094	0.015		0.005	0.0112
		Time	212.202	381.21		1731.49	394.70
	full-batch	RL_2 u	0.041	0.0328		0.0189	0.0184
		RL_2 f	0.0027	0.0089		0.0021	0.0087
		Time	990.80	711.84		2730.96	607.67
Klein Gordon[?]	mini-batch	RL_2 u	0.161	0.0124		0.0154	0.018
		RL_2 f	0.003	0.0033		0.00316	0.00096
		Time	275.93	341.48		1598.275	1141.30
	full-batch	RL_2 u	0.064	0.0594		0.0063	
		RL_2 f	0.002	0.0016		0.002	
		Time	1759.70	2105.53		2370.46	
Time Independent Lid-driven Cavity[?]	mini-batch	RL_2 u	0.097			0.39	0.038
		RL_2 v	0.1687			0.55	0.0524
		Time	1200.01			2692.81	7459.63
	full-batch	RL_2 u	0.05			0.343	0.030434
		RL_2 v	0.0862			0.450	0.04129
		Time	6953.33			12346.34	23525.74
Navier-Stokes[?]	mini-batch	RL_2 u	0.			0.	0.
		RL_2 v	0.			0.	0.
		Time	.0			.0	.0
	full-batch	RL_2 u	0.0			0.0	0.
		RL_2 v	0.0			0.0	0.
		Time	0.0			0.0	.0

For the M_1 model, we used fixed weighting values set with the best empirical settings by giving large weights to the terms with significant generalization errors on unseen test dataset. We used these empirical values as initial values for M_2 and M_3 adaptive weight models for the first 1000 iterations. In M_2, we used formula 5 to adjust the weights of the different terms in the loss function, while in M_3, we assigned different weights to each output of the neural network and used formula 6 to update these weights. To avoid frequent fluctuations of the loss function, the weights are updated every 1000 iteration using 5 and 6 for M_2 and M_3, respectively.

In the following sections, we examine the performance of PINN models discussed above for the solution of selected laminar flow cases described by Navier-Stokes equations and compare results with the numerical solutions. In all cases, the system is governed by incompressible Navier-Stokes equation of the form:

$$
\begin{aligned}
u_t + \mathbf{u}.\nabla\mathbf{u} + \nabla p - \frac{1}{Re}\nabla^2\mathbf{u} = f && \text{in} \quad \Omega \\
\nabla.\mathbf{u} = 0 && \text{in} \quad \Omega \\
\mathbf{u}(\mathbf{0},\mathbf{x}) = g && \text{in} \quad \Omega && \text{Initial Condition} \\
\mathbf{u}(\mathbf{t},\mathbf{x_0}) = h && \text{in} \quad \Gamma_1 && \text{Dirichlet Boundary Condition} \\
\mathbf{u}(\mathbf{t},\mathbf{x_l}) = l && \text{in} \quad \Gamma_0 && \text{Dirichlet Boundary Condition}
\end{aligned}
\tag{7}
$$

where p , f is a scalar pressure field and external force respectively, $\mathbf{u}(\mathbf{t},\mathbf{x}) = (u(t,x), v(t,x))$ is a velocity vector field, and $\mathbf{x} = (x,y) \in \Omega$. We map spatial coordinates (x,y) to the latent solution functions $u(x,y)$, $v(x,y)$ and $p(x,y)$,
Therefore, the momentum and continuity equations are:

$$
\begin{aligned}
r_u &= u_t + (uu_x + vu_y) - \frac{1}{Re}(u_{xx} + u_{yy}) + p_x \\
r_v &= v_t + (uv_x + vv_y) - \frac{1}{Re}(v_{xx} + v_{yy}) + p_y \\
r_c &= u_x + v_y
\end{aligned}
\tag{8}
$$

- **Case 1: classical steady-state flow in a 2D lid-driven cavity:** well-known benchmark problem with one boundary moving with a constant velocity while the other boundaries are fixed.

  The governing equations are:

$$
\begin{aligned}
u_t + \mathbf{u}.\nabla\mathbf{u} + \nabla p - \frac{1}{Re}\nabla^2\mathbf{u} &= 0 && \text{in} \quad \Omega \\
\nabla.\mathbf{u} &= 0 && \text{in} \quad \Omega \\
\mathbf{u}(\mathbf{0},\mathbf{x}) &= g && \text{in} \quad \Omega \\
\mathbf{u}(\mathbf{t},\mathbf{x_0}) &= 0 && \text{in} \quad \Gamma_1 \\
\mathbf{u}(\mathbf{t},\mathbf{x_l}) &= 1 && \text{in} \quad \Gamma_0
\end{aligned}
\tag{9}
$$

  The computation domain Ω is a two-dimensional square cavity $\Omega = (0,1) \times (0,1)$. Uniform discretization is used with grid $(N_x, N_y) = (100, 100)$ and 10 seconds total simulation time with 0.01s time interval. Γ_1 is the top Dirichlet boundary condition of the cavity with velocity tangent to this side, Γ_0 denotes the other three stationary sides, and $Re = 100$ is the Reynolds number of the flow. In this and the fo lowing cases, the finite volume method is used for discretization with OpenFOAM's finite volume C++ library to find the solution.

- **Case 2: 2D blood flow in a tube with no-slip walls** : This problem is a Newtonian, incompressible blood flow simulation in a tube.

  The governing equations are:

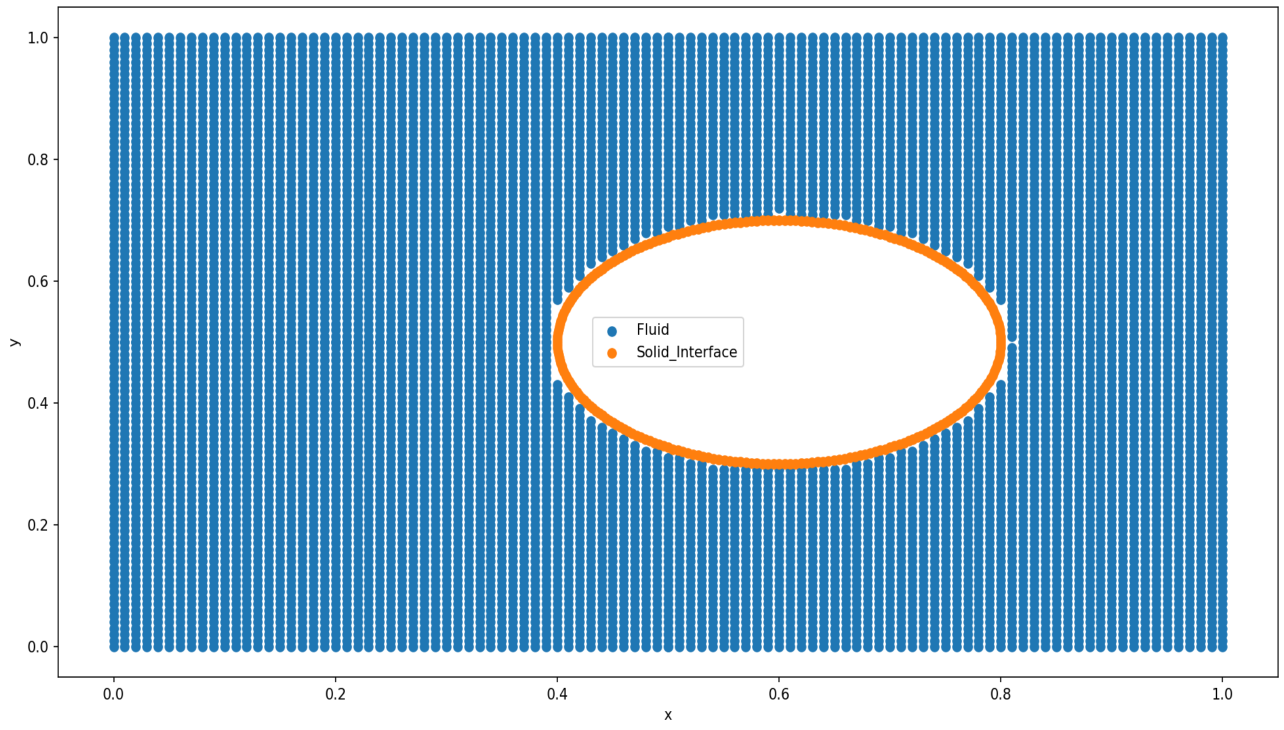
$$
\begin{aligned}
u_t + \mathbf{u}.\nabla\mathbf{u} + \nabla p - \frac{1}{Re}\nabla^2\mathbf{u} &= 0 && \text{in} \quad \Omega \\
\nabla.\mathbf{u} &= 0 && \text{in} \quad \Omega \\
\mathbf{u}(\mathbf{0},\mathbf{x}) &= g && \text{in} \quad \Omega \\
\mathbf{u}(\mathbf{t},\mathbf{x_0}) &= 0 && \text{in} \quad \Gamma_1 \\
\mathbf{u}(\mathbf{t},\mathbf{x_l}) &= 1 && \text{in} \quad \Gamma_0
\end{aligned}
\tag{10}
$$

  where $(t,x,y) \in (0,5) \times (0.0,1) \times (-7.5 * 10^{-3}, 7.5 * 10^{-3})$. Viscosity = $0.00345 m^2/s$ and density = 1056 with initial and inlet velocity 0.2m/s in the x-direction and 0.0m/s in the y-direction. Zero gradient ve city at the outlet in the x and y direction (i.e $\|\frac{\partial u}{\partial x}\|_{x=x_{max}} = \|\frac{\partial v}{\partial x}\|_{x=x_{max}} = 0.0$). Zero gradient pressure at the inlet and on the wall (i.e. $\|\frac{\partial p}{\partial x}\|_{x=0} = \|\frac{\partial p}{\partial y}\|_{y=\pm r} = 0.0$). The initial wall pressure is equal to zero, and wall velocity is always zero ($u_x = u_y = 0$), i.e., no-slip boundary conditions are applied on the walls. The inner fluid domain includes 340000 hexa-cells, while the wall surface has 40000 quad-cells. Cell-center locations are aligned in the x-direction.

- **Case 3: Plane Poiseuille Flow**: a laminar flow between two infinite plates. As opposed to the no-slip case, uniform discretization is used in this case with regular grid $(N_x, N_y) = (62, 2001)$ where $x \in [0, 1]$ and $y \in [-0.0075, 0.0075]$ with 100 time steps for 5-second simulation. The geometry is the same as the pipe flow with boundary conditions, the same as the no-slip case. The inlet velocity is $0.2m/s$, kinematic viscosity is $3.3144 * 10^{-6} m^2 s$ and the density is $1060 kgm^3$.

- **Case 4: Cavity problem with Immersed boundary condition**: This is a direct forcing immersed boundary (IB) problem. Finite difference method is used for the discretization of the fluid with a set of Lagrangian marker points to describe the moving solid boundary. Besides, the finite element method is used to discretise the solid domain. More sepcifically a no-slip boundary condition is enforced on the solid boundary. Figure **??** shows the computational domain at time step 0, where the white colour inside the solid interface represents the solid region.

FIGURE 2 Computational domain of case 4 at time step 0



The governing equations of the fluid are incompressible Navier-Stokes equations with boundary conditions similar to the cavity problem discussed above and volume force added to the momentum equation as follows,

$$\begin{aligned} u_t + \mathbf{u}.\boldsymbol{\nabla}\mathbf{u} + \nabla p - \frac{1}{Re}\boldsymbol{\nabla}^2\mathbf{u} &= f \quad \text{in} \quad \Omega \\ \boldsymbol{\nabla}.\mathbf{u} &= 0 \quad \text{in} \quad \Omega \\ \mathbf{u}(\mathbf{0}, \mathbf{x}) &= g \quad \text{in} \quad \Omega \\ \mathbf{u}(\mathbf{t}, \mathbf{x_0}) &= 0 \quad \text{in} \quad \Gamma_1 \\ \mathbf{u}(\mathbf{t}, \mathbf{x_1}) &= 1 \quad \text{in} \quad \Gamma_0 \\ f &= \int_s \mathbf{F}(\xi(s,t))\delta(\xi - \mathbf{x})ds, \text{transfer between surface and volume forces} \end{aligned} \tag{11}$$

Where p, f is a scalar pressure field and volume force, respectively. F is the single orce that is added to the fluid-solid interface, $\mathbf{u(t, x)} = (u(t,x), v(t,x))$ is a velocity vector field, and $\mathbf{x} = (x, y) \in \Omega_{fluid}$ computational domain excluding the immersed object, ξ is the embedded fluid-solid interface, s is the coordinate along the interface.

The experiments are performed on a cluster with Intel XEON 8362 processors, with 512GB total memory, NVIDIA A100 80 GB PCIe and HDR InfiniBand (200 Gbps). Different PINN configurations are examined to get the best results. For a better convegence rate, the input variables are standardized with mean zero and standard deviation of one in all cases. All codes are implmented using the TensorFlow library. Although results a e not reported here, mean squares error loss works better than L_1 loss in all cases. Therefore, mean sq ares error loss is used in all cases.

For numerical solution, TODO.

Training datasets are selected using the Sobol distribution and ordered by time before training.

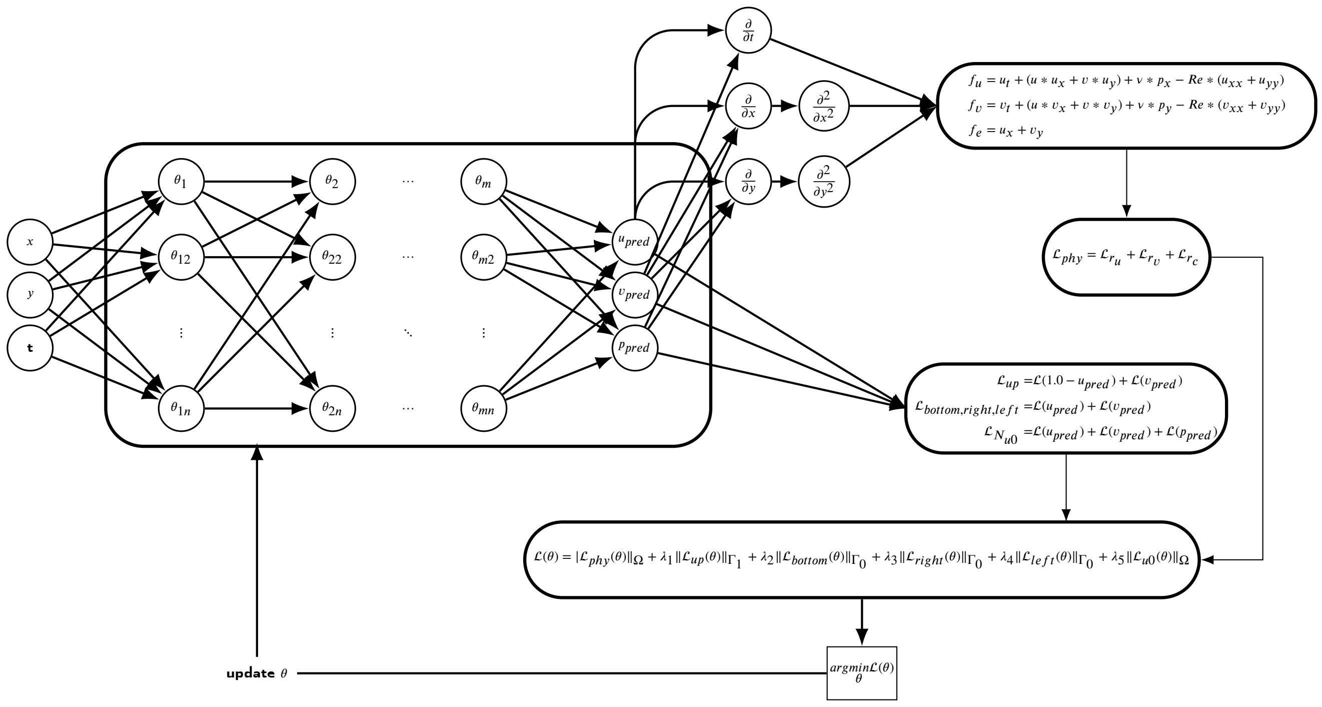
2.1 | Case1: 2D Flow in a lid-driven cavity

The loss function for solving the problem using fixed weighting is:

$$\mathcal{L}(\theta) = |\mathcal{L}_{phy}(\theta)\|_{\Omega} + \lambda_1\|\mathcal{L}_{up}(\theta)\|_{\Gamma_1} + \lambda_2\|\mathcal{L}_{bottom}(\theta)\|_{\Gamma_0} + \lambda_3\|\mathcal{L}_{right}(\theta)\|_{\Gamma_0} + \lambda_4\|\mathcal{L}_{left}(\theta)\|_{\Gamma_0} + \lambda_5\|\mathcal{L}_{u0}(\theta)\|_{\Omega} \tag{12}$$

$$\begin{aligned} \mathcal{L}_{phy} &= \mathcal{L}_{r_u} + \mathcal{L}_{r_v} + \mathcal{L}_{r_c} \\ \mathcal{L}_{up} &= \mathcal{L}(1.0 - u_{\text{pred}}) + \mathcal{L}(v_{\text{pred}}) \\ \mathcal{L}_{bottom,right,left} &= \mathcal{L}(u_{\text{pred}}) + \mathcal{L}(v_{\text{pred}}) \\ \mathcal{L}_{N_{u0}} &= \mathcal{L}(u_{\text{pred}}) + \mathcal{L}(v_{pred}) + \mathcal{L}(p_{pred}) \end{aligned} \tag{13}$$

We do not consider pressure values during training except at the initial condition.



x
y
t
θ_1
θ_{12}
θ_{1n}
θ_2
θ_{22}
θ_{2n}
θ_m
θ_{m2}
θ_{mn}
u_{pred}
v_{pred}
p_{pred}
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial}{\partial y}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial^2}{\partial y^2}$
$f_u = u_t + (u * u_x + v * u_y) + v * p_x - Re * (u_{xx} + u_{yy})$
$f_v = v_t + (u * v_x + v * v_y) + v * p_y - Re * (v_{xx} + v_{yy})$
$f_e = u_x + v_y$
$\mathcal{L}_{phy} = \mathcal{L}_{r_u} + \mathcal{L}_{r_v} + \mathcal{L}_{r_c}$
$\mathcal{L}_{up} = \mathcal{L}(1.0 - u_{pred}) + \mathcal{L}(v_{pred})$
$\mathcal{L}_{bottom,right,left} = \mathcal{L}(u_{pred}) + \mathcal{L}(v_{pred})$
$\mathcal{L}_{N_{u0}} = \mathcal{L}(u_{pred}) + \mathcal{L}(v_{pred}) + \mathcal{L}(p_{pred})$
$\mathcal{L}(\theta) = |\mathcal{L}_{phy}(\theta)\|_{\Omega} + \lambda_1\|\mathcal{L}_{up}(\theta)\|_{\Gamma_1} + \lambda_2\|\mathcal{L}_{bottom}(\theta)\|_{\Gamma_0} + \lambda_3\|\mathcal{L}_{right}(\theta)\|_{\Gamma_0} + \lambda_4\|\mathcal{L}_{left}(\theta)\|_{\Gamma_0} + \lambda_5\|\mathcal{L}_{u0}(\theta)\|_{\Omega}$
$\underset{\theta}{argmin}\mathcal{L}(\theta)$
update θ

FIGURE 3 Schematic of PINN model

Training Details

We train for 40,000 iterations on the regular grid datasets and learning rate with initial value $5 * 10^{-4}$ and exponential decay with decay step 1000 and decay rate 0.9. We examined $tanh()$, $xsig(x)$ and $hard - swish(x)$ as non-linear activation functions. However, $xsig(x)$ gives better results than other activation functions using these settings. Full-batch training shows better results

TABLE 5 case 1: 2D lid-driven cavity problem solution result using different settings where RelL_2 = $\sqrt[2]{\frac{\sum(exact-pred)^2}{\sum(exact)^2}} *$ 100% using xsigmoid activation function and Sobol distribution for training data selection. Time is the total time in Hours:Minutes:Seconds.

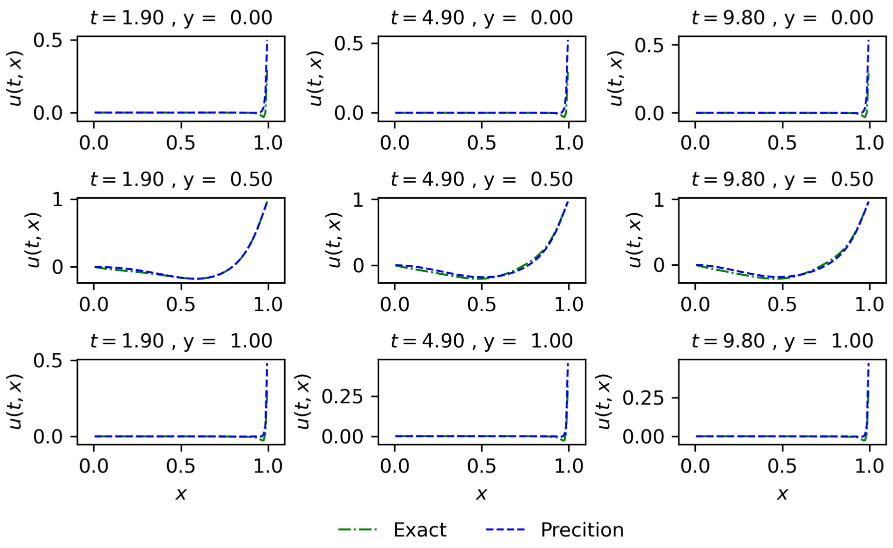
Model	u	v	p	Time(HH:MM:SS)
M_1	$5.825e+00$	$5.145e+00$	$2.229e+01$	17:00:04
M_2	$4.726e+00$	$3.682e+00$	$2.055e+01$	19:12:31
M_3	$3.924e+00$	$2.311e+00$	$8.890e+00$	22:04:27
M_4	$5.629e+00$	$4.246e+00$	$2.234e+01$	7:04:34

than mini-batch training with Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ followed by full-batch LBFGS-B method. Fully connected feedforward neural network with $[3] + 4 * [256] + [3]$, i.e 10 layers with 50 neurons per layer besides the input and output layers. Experimental results also show that accuracy increases when only velocity components are used for training on the boundary, so we ignore pressure values on the boundary. Finally, the total training dataset is 12822 (1.22% from the total dataset).

Results are shown in table 5. We draw the attention on that values of the M_1 model are used as initial values for the first 1000 iterations in the M_2 and M_3 models. We also used M_2 for each loss term in the objective function, while the M_3 model is used for each neural network output. According to these results, the M_2 model performs slightly better than other models.

Figures 4, 5 and 6 show the U V velocity components and pressure respectively in the X-direction versus time for different Y values for model M_2 in table 5.

FIGURE 4 CFD U velocity vs PINN prediction solution for different time and y values using M_2 model with relative L_2 error $3.924e+00\%$



Figures 7 shows the prediction solution at timestep 100, which is the steady state for U-velocity(left), V-velocity (centre) and pressure (right) for PINN(up), CFD(middle) and the absolute error (bottom). The figure shows a good agreement between the PINN and CFD solutions.

FIGURE 5 CFD V velocity vs PINN prediction solution for different time and y values using M_2 model with relative L_2 error $2.311e+00\%$

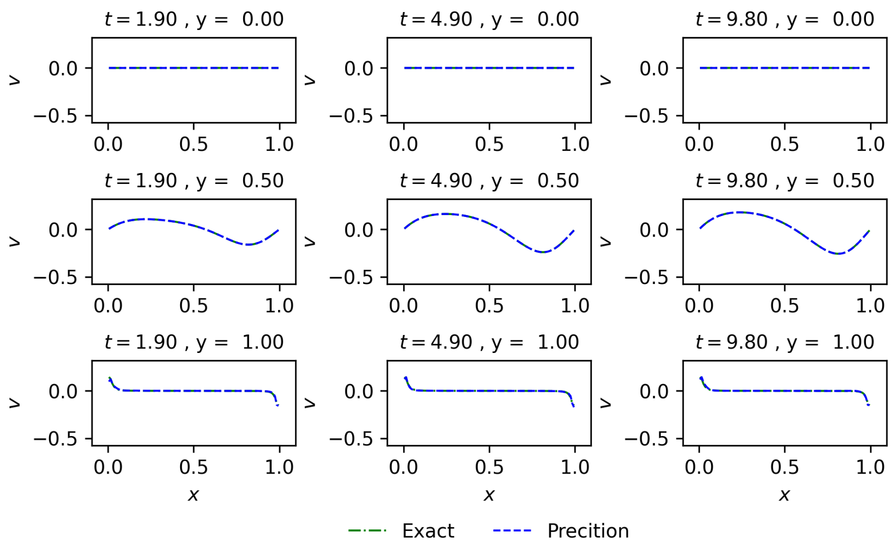


FIGURE 6 CFD pressure vs PINN prediction solution for different time and y values using M_2 model with relative L_2 error $8.890e+00\%$

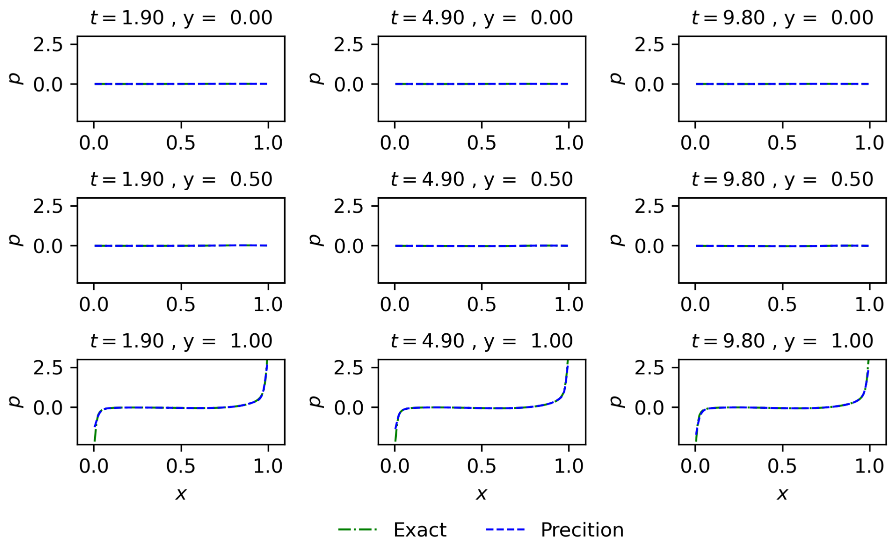
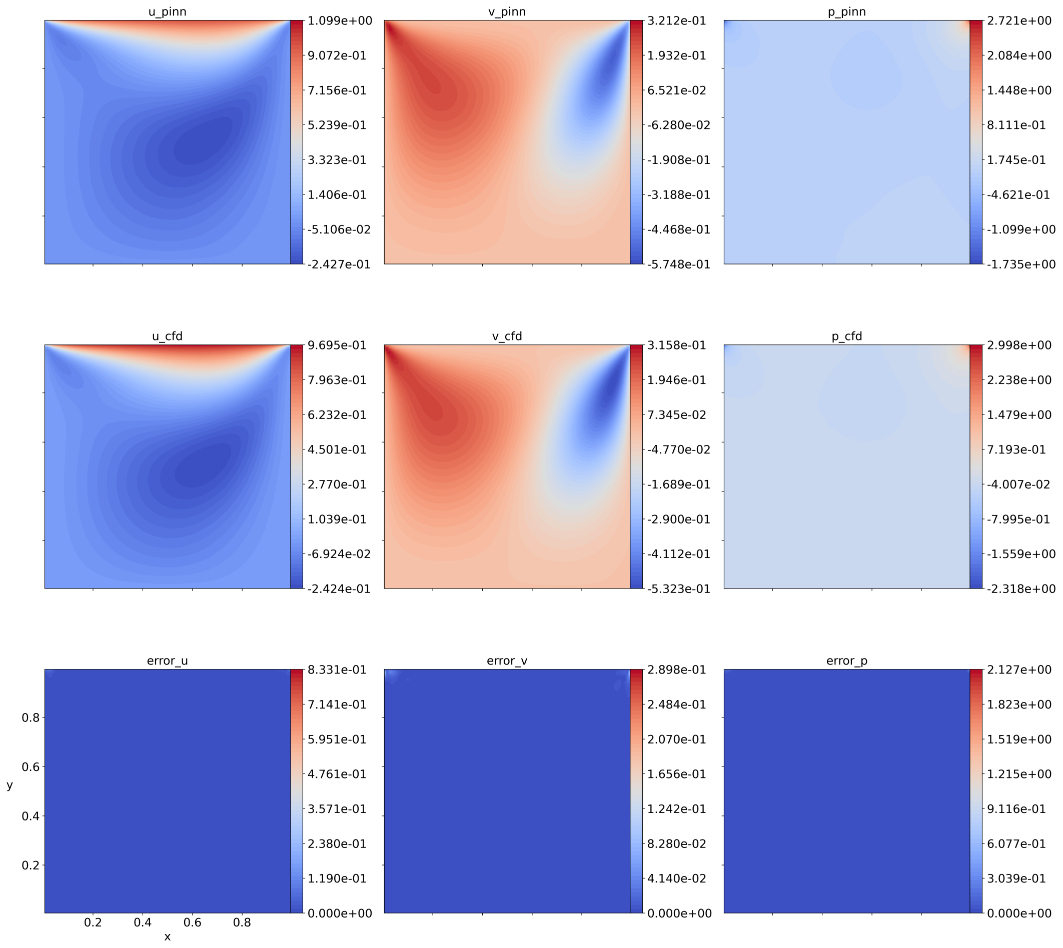


FIGURE 7 prediction solution at the steady state timestep 100 for U-velocity(left), V-velocity (centre) and pressure (right) for PINN(up), CFD(middle) and the absolute error (bottom) using M_2 model with Rel L_2 error equals to 9.542%, $1.778e+01$% and $3.462e+01$% for U, V and P respectively.



2.2 | Case 3: 2D blood flow in a tube with slip boundary condition

The loss function for solving the problem is:

$$
\begin{aligned}
\mathcal{L}(\theta) :=& \lambda_1 \|\mathcal{L}_s(\theta)\|_\Omega + \lambda_2 \|\mathcal{L}_{\text{wall}}(\theta)\| + \lambda_3 \|\mathcal{L}_{\text{inlet}}(\theta)\| + \lambda_4 \|\mathcal{L}_{\text{outlet}}(\theta)\| + \\
& \lambda_5 \|\mathcal{L}_{\text{initial}}(\theta)\| \\
\mathcal{L}_{\text{s}} =& \mathcal{L}(u - u_{\text{pred}}) + \mathcal{L}(v - v_{\text{pred}}) + \mathcal{L}(p - p_{\text{pred}}) \\
\mathcal{L}_{\text{wall}} =& \mathcal{L}(0.4 - u_{\text{pred}}) + \mathcal{L}(v - v_{\text{pred}}) + \mathcal{L}(p - p_{\text{pred}}) + \mathcal{L}(\frac{\partial p_{\text{pred}}}{\partial y}) \\
\mathcal{L}_{\text{inlet}} =& \mathcal{L}(0.4 - u_{\text{pred}}) + \mathcal{L}(v_{\text{pred}}) + \mathcal{L}(p - p_{\text{pred}}) + \mathcal{L}(\frac{\partial p_{\text{pred}}}{\partial x}) \\
\mathcal{L}_{\text{outlet}} =& \mathcal{L}(u - u_{\text{pred}}) + \mathcal{L}(\frac{\partial u_{\text{pred}}}{\partial x}) + \mathcal{L}(v - v_{\text{pred}}) + \mathcal{L}(\frac{\partial v_{\text{pred}}}{\partial x}) + \mathcal{L}(p_{\text{pred}}) \\
\mathcal{L}_{\text{initial}} =& \mathcal{L}(0.4 - u_{\text{pred}}) + \mathcal{L}(v_{\text{pred}}) + \mathcal{L}(p_{\text{pred}})
\end{aligned}
\tag{14}
$$

Training Details

For the slip and no-slip cases. In our experiments we observe that stochastic gradient descent with batch size of 300 and 40000 iteration per epoch works better than the full-batch approach with Adam optimizer and $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ with initial learning rate value $5 * 10^{-3}$ and exponential decay with decay steps 3000 and decay rate 0.9. The network is $[3 + 3 * [50] + [3]$. Sobol sequence is used for training data selection with hard-swish as the activation function, which performs better than $tanh$ and $xsig$.

Experimental results also show that adding PDE physical loss along with the boundary loss on the same neural network degrades performance and that using only supervised learning on the boundary without PDE loss or putting PDE loss on a separate neural network gives better results. Therefore, we just use supervised learning on the boundary and initial training dataset. We also randomly selected 125 collocation points from the domain and trained them supervised to enhance results as shown in equation 14. Hence the trained model can not be generalized outside the boundary dataset. M_2 and M_3 models can not be used in this case as they rely on residual loss for adjusting weights.

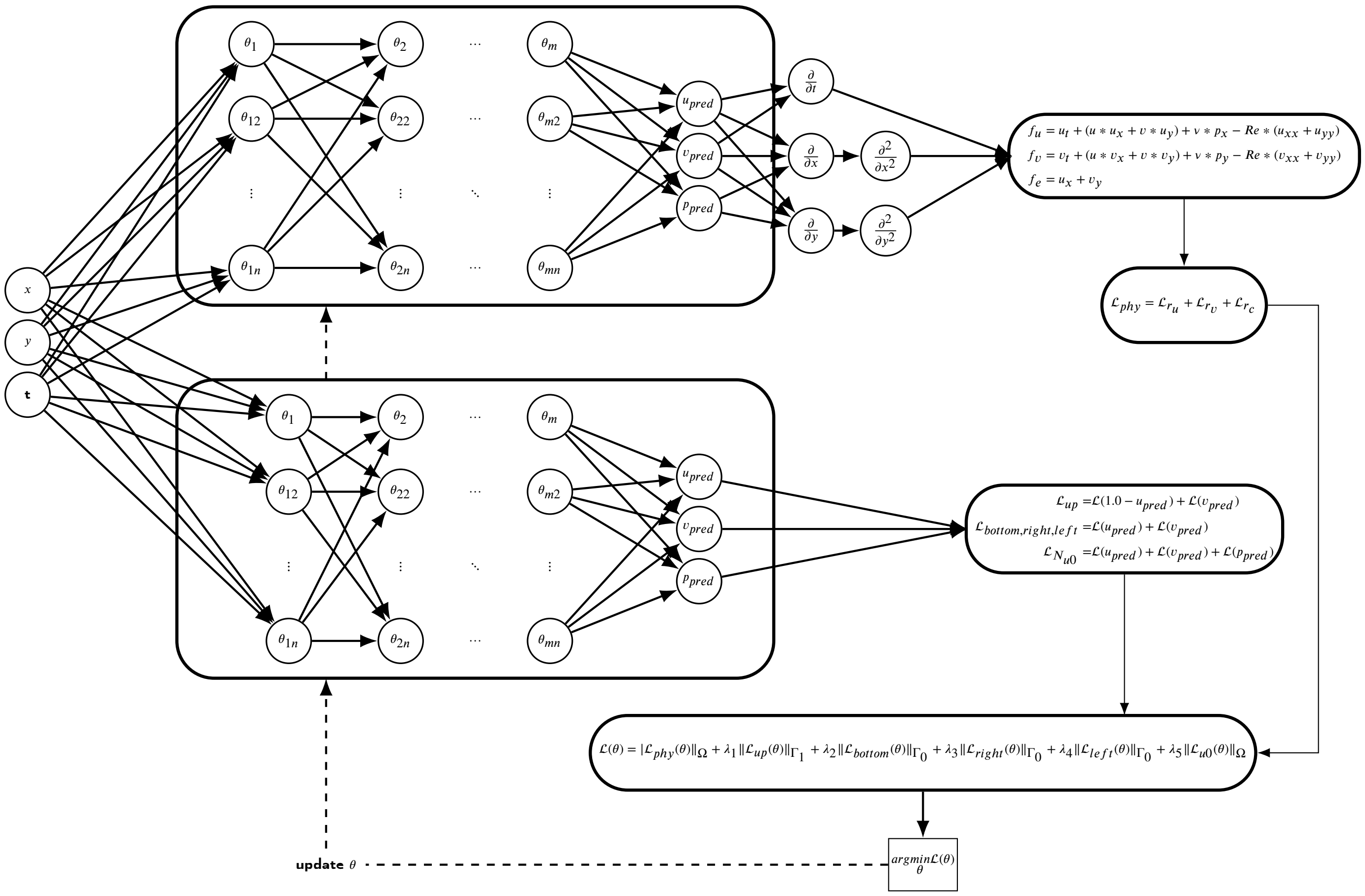
2.3 | Case 2: 2D blood flow in a tube with no-slip walls

The loss function for solving the problem is:

TABLE 6 Case3: 2D blood flow in a tube with slip boundary condition approximation results using different settings where $\text{Rel}L_p = \sqrt[p]{\frac{\sum(exact-pred)^p}{\sum(exact)^p}} * 100\%$. Sobol sequence is used for training data selection with hard-swish as the activation function. TTT is the total t sting time in seconds.

Model	Error(%)	u	v	p	TTT(S)
M1	L_2	$4.963e-02$	$1.001e+02$	$2.147e+02$	3816
	L_1	$4.277e-02$	$1.001e+02$	$1.986e+02$	

$$
\begin{aligned}
\mathcal{L}(\theta) :=& \lambda_1 \|\mathcal{L}_s(\theta)\|_\Omega + \lambda_2 \|\mathcal{L}_{\text{wall}}(\theta)\| + \lambda_3 \|\mathcal{L}_{\text{inlet}}(\theta)\| + \lambda_4 \|\mathcal{L}_{\text{outlet}}(\theta)\| + \\
& \lambda_5 \|\mathcal{L}_{\text{initial}}(\theta)\| \\
\mathcal{L}_{\text{s}} =& 50 * \mathcal{L}(u - u_{\text{pred}}) + 2000 * \mathcal{L}(v - v_{\text{pred}}) + 1000 * \mathcal{L}(p - p_{\text{pred}}) \\
\mathcal{L}_{\text{wall}} =& 50 * \mathcal{L}(u_{\text{pred}}) + 2000 * \mathcal{L}(v_{\text{pred}}) + 1000\big(\mathcal{L}(p - p_{\text{pred}}) + \mathcal{L}(\frac{\partial p_{\text{pred}}}{\partial y})\big) \\
\mathcal{L}_{\text{inlet}} =& 50 * \mathcal{L}(0.2 - u_{\text{pred}}) + 2000 * \mathcal{L}(v_{\text{pred}}) + 1000 * \big(\mathcal{L}(p - p_{\text{pred}}) + \mathcal{L}(\frac{\partial p_{\text{pred}}}{\partial x})\big) \\
\mathcal{L}_{\text{outlet}} =& 50 * \big(\mathcal{L}(u - u_{\text{pred}}) + \mathcal{L}(\frac{\partial u_{\text{pred}}}{\partial x})\big) + 2000 * \big(\mathcal{L}(v - v_{\text{pred}}) + \mathcal{L}(\frac{\partial v_{\text{pred}}}{\partial x}) + \mathcal{L}(p_{\text{pred}})\big) \\
\mathcal{L}_{\text{initial}} =& 50 * \mathcal{L}(0.2 - u_{\text{pred}}) + 2000 * \mathcal{L}(v_{\text{pred}}) + 1000 * \mathcal{L}(p_{\text{pred}})
\end{aligned}
\tag{15}
$$



x
y
t
θ_1
θ_{12}
θ_{1n}
θ_2
θ_{22}
θ_{2n}
θ_m
θ_{m2}
θ_{mn}
u_{pred}
v_{pred}
p_{pred}
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial}{\partial y}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial^2}{\partial y^2}$
$f_u = u_t + (u * u_x + v * u_y) + \nu * p_x - Re * (u_{xx} + u_{yy})$
$f_v = v_t + (u * v_x + v * v_y) + \nu * p_y - Re * (v_{xx} + v_{yy})$
$f_e = u_x + v_y$
$\mathcal{L}_{phy} = \mathcal{L}_{r_u} + \mathcal{L}_{r_v} + \mathcal{L}_{r_c}$
$\mathcal{L}_{up} = \mathcal{L}(1.0 - u_{pred}) + \mathcal{L}(v_{pred})$
$\mathcal{L}_{bottom,right,left} = \mathcal{L}(u_{pred}) + \mathcal{L}(v_{pred})$
$\mathcal{L}_{N_{u0}} = \mathcal{L}(u_{pred}) + \mathcal{L}(v_{pred}) + \mathcal{L}(p_{pred})$
$\mathcal{L}(\theta) = |\mathcal{L}_{phy}(\theta)\|_\Omega + \lambda_1 \|\mathcal{L}_{up}(\theta)\|_{\Gamma_1} + \lambda_2 \|\mathcal{L}_{bottom}(\theta)\|_{\Gamma_0} + \lambda_3 \|\mathcal{L}_{right}(\theta)\|_{\Gamma_0} + \lambda_4 \|\mathcal{L}_{left}(\theta)\|_{\Gamma_0} + \lambda_5 \|\mathcal{L}_{u0}(\theta)\|_\Omega$
$\underset{\theta}{argmin} \mathcal{L}(\theta)$
update θ

FIGURE 8 Schematic of PINN model

Training Details

We use irregular grid datasets and learning rates exponential decay with initial value $5 * 10^{-3}$. We use Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. The network is [3]+3 ∗ [50]+[3]. Here, we also use supervised learning without the PDE loss in the objective function. Sobol sequence is used for training data selection with hard-swish as the activation function, which performs better than $tanh$ and $xsig$—total training dataset size: 2860 (0.10%).

Experimental results also show that adding PDE physical loss along with the boundary loss on the same neural network degrades performance and that using only supervised learning on the boundary without PDE loss. Therefore, we use supervised

TABLE 7 Case4: 2D blood flow in a tube with no-slip walls boundary condition approximation results using different settings where Abs$L_2 = \sqrt[2]{\frac{1}{N}\sum(exact - pred)^2} * 100\%$, Rel$L_2 = \sqrt[2]{\frac{\sum(exact-pred)^2}{\sum(exact)^2}} * 100\%$. Time is the total time in Hours:Minutes:Seconds.

Model	Error	u	v	p	Time(HH:MM:SS)
M_1	L_2	$4.718e+00$	$7.456e+01$	$1.608e+00$	10:43:17
	L_1				
M_2	L_2	$4.400e+01$	$9.570e+01$	$4.081e+01$	13:01:10
	L_1				
M_3	L_2	$2.951e+01$	$9.856e+01$	$4.025e+01$	15:22:13
	L_1				
M_4	L_2	$3.058e+00$	$8.083e+01$	$3.173e+01$	2:33:01
	L_1				

TABLE 8 Case 5: 2D Poiseuille problem approximation results using different settings where Rel$L_p = \sqrt[p]{\frac{\sum(exact-pred)^p}{\sum(exact)^p}} *$ 100%. Sobol sequence is used for training data selection with hard-swish as the activation function. Time is the total time in Hours:Minutes:Seconds.

Model	u	v	p	Time(HH:MM:SS)
M1	$6.352e+00$	$9.846e+01$	$3.701e+01$	1:10:24
M2	$2.063e+01$	$1.000e+02$	$4.242e+01$	2:10:23
M3	$1.792e+01$	$1.000e+02$	$4.533e+01$	2:14:15
M_4				

learning on the boundary and initial training dataset. We also randomly selected 780 collocation points from the domain and trained them in a supervised way to enhance results as shown in equation 15. Hence, the trained model can not be generalized outside the boundary dataset.

2.4 | Case 3: 2D Poiseuille problem

The loss function is the same as the loss function of the no slipcase of equation 15

Training Details

The network is $[3] + 3 * [50] + [3]$ with exponential decay learning rates with initial value 10^{-3}. Stochastic gradient descent works better than full-batch training with Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Sobol sequence is used for training data selection with hard-swish as the activation function. Here we also use supervised learning without the PDE loss in the objective function. Total training dataset size: 7007 (0.056% of the total dataset)

Figures 18 shows the prediction solution at timestep 100, which is the steady state for U-velocity(left), V-velocity (centre) and pressure (right) for PINN(up), CFD(middle) and the absolute error (bottom). The figure shows a good agreement between the PINN and CFD solutions.

FIGURE 9 CFD U velocity vs PINN prediction solution the 2D Poiseuille problem at different time and y values using M_1 model with relative L_2 error $6.352e+00\%$

FIGURE 10 CFD V velocity vs PINN prediction solution the 2D Poiseuille problem at different time and y values using M_1 model with relative L_2 error $9.846e+01\%$

FIGURE 11 CFD pressure vs PINN prediction solution the 2D Poiseuille problem at different time and y values using M_1 model with relative L_2 error $3.701e+01\%$

FIGURE 12 prediction solution for 2D Poiseuille problem case 5 at the steady state timestep 100 for U-velocity(left), V-velocity (centre) and pressure (right) for PINN(up), cfd(middle) and the absolute error (bottom) using M_1 model with Rel L_2 error equals to $1.123e+01\%$, $9.903e+01\%$ and $4.269e+01\%$ for U, V and P respectively.

2.5 | Case4: Cavity problem with moving Immersed boundary condition

The loss function for solving the problem using fixed weighting is:

$$\begin{aligned}\mathcal{L}(\theta) :=&\lambda_1\|\mathcal{L}^{fluid}_{phy}(\theta)\|_{\Omega_f} + \lambda_2\|\mathcal{L}_s(\theta)\|_{\Omega_{solid}} + \lambda_3\|\mathcal{L}_{SI}(\theta)\|_{\Gamma_{SI}} + \lambda_4\|\mathcal{L}_{up}(\theta)\|_{\Gamma_1} + \lambda_5\|\mathcal{L}_{\text{bottom}}(\theta)\|_{\Gamma_0} + \lambda_6\|\mathcal{L}_{\text{left}}(\theta)\|_{\Gamma_0}+\\ &\lambda_7\|\mathcal{L}_{\text{right}}(\theta)\|_{\Gamma_0} + \lambda_8\|\mathcal{L}_{u0}(\theta)\|_{\Omega_f}\end{aligned} \tag{16}$$

$$\begin{aligned}\mathcal{L}(\theta) :=&\|\mathcal{L}^{fluid}_{phy}(\theta)\|_{\Omega_f} + \|\mathcal{L}_s(\theta)\|_{\Omega} + \|\mathcal{L}_{SI}(\theta)\|_{\Gamma_{SI}} + 2.0 * \Big(\|\mathcal{L}_{up}(\theta)\|_{\Gamma_1} + \|\mathcal{L}_{\text{bottom}}(\theta)\|_{\Gamma_0} + \|\mathcal{L}_{\text{left}}(\theta)\|_{\Gamma_0}+\\ &\lambda_5\|\mathcal{L}_{\text{right}}(\theta)\|_{\Gamma_0} + 2.0 * \|\mathcal{L}_{u0}(\theta)\|_{\Omega_f}\Big)\\ \mathcal{L}^{fluid}_{phy} =&\mathcal{L}_{r_u} + \mathcal{L}_{r_v} + \mathcal{L}_{r_c}\\ \mathcal{L}_s =&\mathcal{L}(u-u_{\text{pred}}) + \mathcal{L}(v-v_{\text{pred}}) + \mathcal{L}(p-p_{\text{pred}}) + \mathcal{L}(f_x - f_{x_{\text{pred}}}) + \mathcal{L}(f_y - f_{y_{\text{pred}}})\\ \mathcal{L}_{SI} =&\mathcal{L}(u-u_{\text{pred}}) + \mathcal{L}(v-v_{\text{pred}}) + \mathcal{L}(p-p_{\text{pred}}) + \mathcal{L}(f_x - f_{x_{\text{pred}}}) + \mathcal{L}(f_y - f_{y_{\text{pred}}})\\ \mathcal{L}_{up} =&\mathcal{L}(1.0-u_{\text{pred}}) + \mathcal{L}(v_{\text{pred}})\\ \mathcal{L}_{bottom,right,left} =&\mathcal{L}(u_{\text{pred}}) + \mathcal{L}(v_{\text{pred}})\\ \mathcal{L}_{u_0} =&\mathcal{L}(u_{\text{pred}}) + \mathcal{L}(v_{\text{pred}}) + \mathcal{L}(p_{\text{pred}})\end{aligned} \tag{17}$$

Where L^{fluid}_{phy} is the physics loss of the fluid region, L_s, $\mathcal{L}_{SI}$ are supervised loss corresponding to some sensor points selected from the solid domain Ω_{solid} and solid interface Γ_{SI} respectively. $\mathcal{L}_{left}$, $\mathcal{L}_{right}$, $\mathcal{L}_{up}$, $\mathcal{L}_{bottom}$, $\mathcal{L}_{u_0}$, are the left, right, up, bottom and initial loss respectively.

Training Details

We train for 40,000 iterations on the regular grid datasets and learning rate with initial value $5 * 10^{-4}$ and exponential decay with decay step 1000 and decay rate 0.9. We use $xsig(x)$ non-linear activation function. Full-batch training shows better results than mini-batch training with Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ followed by full-batch LBFGS-B method. Fully connected f dforward neural network with [3] + 4 ∗ [256] + [5], i.e 4 layers with 256 neurons per layer besides the input and output layers. The input variables are $\{t, x, y\}$ and the output variables are $\{u_{\text{pred}}, v_{\text{pred}}, p_{\text{pred}}, f_{x_{\text{pred}}}, f_{y_{\text{pred}}}\}$. Only velocity components are used for training on the fluid boundary and recover pressure and the force component values as latent variables. Finally, the total training dataset is 12822 (1.22% from the total dataset). Training data instances are removed from the testing dataset.

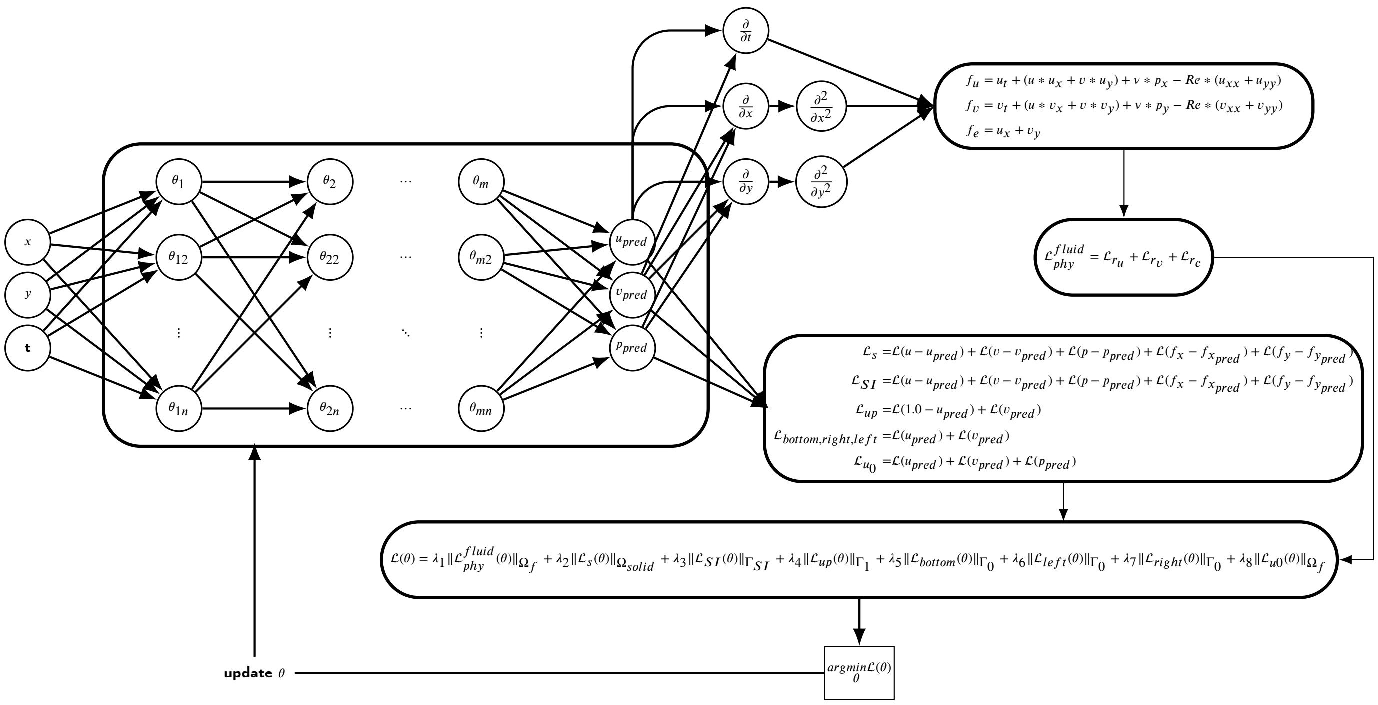


FIGURE 13 Schematic of PINN model

FIGURE 14 CFD U velocity vs PINN prediction for Cavity problem with moving Immersed boundary condition at different time and y values using M_1 model with relative L_2 error $8.578e+00\%$

FIGURE 15 CFD V velocity vs PINN prediction for Cavity problem with moving Immersed boundary condition at different time and y values using M_1 model with relative L_2 error $8.650e+00\%$

TABLE 9 Case 4: Cavity problem with moving Immersed boundary condition approximation results using different settings where $\text{Rel}L_p = \sqrt[p]{\frac{\sum(exact-pred)^p}{\sum(exact)^p}} * 100\%$. Time is the total time in Hours:Minutes:Seconds.

Model	Error	u	v	p	f_x	f_y	Time(DAY:HH:MM:SS)
M1	RL_2	$1.316e+01$	$2.133e+01$	$2.893e+01$	$6.282e+01$	$6.088e+01$	15:50:17
	AL_2						
M2	RL_2	$8.578e+00$	$8.650e+00$	$2.615e+01$	$6.464e+01$	$7.104e+01$	1:08:57:31
	AL_2						
M3	RL_2	$1.508e+01$	$1.662e+01$	$3.239e+01$	$6.978e+01$	$6.675e+01$	1:05:48:29
	AL_2						
M_4	RL_2	$1.343e+01$	$2.206e+01$	$3.671e+01$	$6.059e+01$	$6.084e+01$	18:49:36
	AL_2						

FIGURE 16 CFD pressure vs PINN prediction for Cavity problem with moving Immersed boundary condition at different time and y values using M_1 model with relative L_2 error $6.464e + 01\%$

FIGURE 17 CFD pressure vs PINN prediction solution for Cavity problem with moving Immersed boundary condition at different time and y values using M_1 model with relative L_2 error $7.104e + 01\%$

FIGURE 18 prediction solution for Cavity problem with moving Immersed boundary condition case 5 at the steady state timestep 100 for U-velocity(left), V-velocity (centre) and pressure (right) for PINN(up), cfd(middle) and the absolute error (bottom) using M_1 model with Rel L_2 error equals to $1.123e+01\%$, $9.903e+01\%$ and $4.269e+01\%$ for U, V and P respectively.

3 | DISCUSSION

This work studies the adaptive weighting PINN methods for solving selected laminar flow cases modelled by Navier-Stokes equations. We examined the PI N solution for predicting the velocities in the x and y directions and predicting the pressure and force values as latent variables with different boundary conditions. We also investigate the performance of PINN for solving an immersed boundary problem.

In fixed weighting, we give larger values to the terms with more significant testing errors. Two adaptive methods for adjusting the weights of the different terms of the PINN loss function are examined. In the first model, we call it M_2; the weights are updated according to gradient information using formula 5; in the second model, M_3, the weights are updated according to an empirical NTK matrix of that term using formula 6. Both methods rely on hyperparameters that can affect the performance of the PINN model. Experimental results show that adapting weights using the M_3 model is slightly better than the other models when the physical loss is significant, like the cavity problem in case 1 and the immersed boundary problem in case 4. Proper initialization of weights is essential even with these adaptive methods.

Frequent updates of the weights can lead to fluctuations in the loss function during training. Therefore, we updated weights at every 1000 iterations.

ACKNOWLEDGEMENTS

Computing resources used in this work were provided by the National Center for High-Performance Computing of Turkey (UHeM) under grant number 4005142018.

CONFLICT OF INTEREST

There is no conflict of interest.

References

1. Lu L, Jin P, and Karniadakis GE. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. arXiv preprint arXiv:191003193. 2019;.

2. Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A, et al. Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:201008895. 2020;.

3. Li Z, Zheng H, Kovachki N, Jin D, Chen H, Liu B, et al. Physics-informed neural operator for learning partial differential equations. arXiv preprint arXiv:211103794. 2021;.

4. Raissi M, Perdikaris P, and Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics. 2019;**378**:686–707.

5. Krishnapriyan A, Gholami A, Zhe S, Kirby R, and Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems. 2021;**34**:26548–26560.

6. Wang S, Yu X, and Perdikaris P. When and why PINNs fail to train: A neural tangent kernel perspective. Journal of Computational Physics. 2022;**449**:110768.

7. Wight CL, and Zhao J. Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. arXiv preprint arXiv:200704542. 2020;.

8. Wang S, Sankaran S, and Perdikaris P. Respecting causality is all you need for training physics-informed neural networks. arXiv preprint arXiv:220307404. 2022;.

9. Wang S, Teng Y, and Perdikaris P. Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. SIAM Journal on Scientific Computing. 2021;**43**(5):A3055–A3081.

10. Rahaman N, Baratin A, Arpit D, Draxler F, Lin M, Hamprecht F, et al. On the spectral bias of neural networks. In: International Conference on Machine Learning. PMLR; 2019. p. 5301–5310.

11. Deshpande M, Agarwal S, and Bhattacharya AK. Investigations on convergence behaviour of Physics Informed Neural Networks across spectral ranges and derivative orders. In: 2022 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE; 2022. p. 1172–1179.

12. McClenny L, and Braga-Neto U. Self-adaptive physics-informed neural networks using a soft attention mechanism. arXiv preprint arXiv:200904544. 2020;.

13. Li S, and Feng X. Dynamic Weight Strategy of Physics-Informed Neural Networks for the 2D Navier–Stokes Equations. Entropy. 2022;**24**(9):1254.

14. Jagtap AD, Kawaguchi K, and Karniadakis GE. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. Journal of Computational Physics. 2020;**404**:109136.

15. Ji W, Qiu W, Shi Z, Pan S, and Deng S. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. The Journal of Physical Chemistry A. 2021;**125**(36):8098–8106.

16. Sahli Costabal F, Yang Y, Perdikaris P, Hurtado DE, and Kuhl E. Physics-informed neural networks for cardiac activation mapping. Frontiers in Physics. 2020;**8**:42.

17. Moseley B, Markham A, and Nissen-Meyer T. Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. Advances in Computational Mathematics. 2023;**49**(4):62.

□