Physics-Informed Neural Networks

B.Tech Project Phase-1

N V Sai Gangadhar

Guide:
Asim Tewari
Mechanical Engineering, IIT Bombay

Co-Guide:
Mayank Baranwal
SysCon, IIT Bombay

Presentation Overview

1. Introduction
2. Literature Review
3. Adaptive Sampling and its Variations
4. Experiments
5. Conclusions
6. Future Work

Introduction

- In this Project, we aim to develop novel methods for solving complex PDEs using neural networks as the inference function
- In analysing complex physical systems and engineering systems, we are faced with a challenge of drawing conclusions and decision making based on small data regime, the vast majority of SOTA ML techniques are lacking robustness and fail to provide any guarantee on convergence
- One of the key advantage of PINNs is inferring PDE coefficients from the small set of data available and solving PDE simultaneously
- Another key advantage of PINNs is the lower computational time required for evaluating the function at a point compared to traditional numerical solvers

Terminology

We focus on scientific systems that have a PDE constraint of the following form:

$$u_t + N[u; \lambda] = 0,\ x \in \mathbf{\Omega},\ t \in [0,T]$$

- where u(t,x) is denotes the latent (hidden) solution that we are interested in solving
- N[u; λ] is a non-linear operator parametrized by λ

A little background on 1D Burger's Equation:

All the experiments done on 1D Burgers equation considers the following details:

- $$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], t \in [0, 1],$$
  $$u(x, 0) = -\sin(\pi x),$$
  $$u(-1, t) = u(1, t) = 0,$$
- N_u = Number of supervised training samples; N_f = Number of Collocation Points

Literature Review

1. M Raissi Paper - Simple Implementation of PINN
2. Self-supervising training with adaptive collocation points
3. Theory of functional connections
4. Hyper PINNs
5. An exhaustive review of various sampling methods for PINNs

   (this paper was made available on Oct 22, the phase during which we were working on the same)

Physics-Informed Neural Networks: By M Raissi

Physics-Informed Neural Networks: by M Raissi

- A PINN model for solving Burgers 1D equation has been proposed
- A network with 8 hidden layers and 20 neurons each has been used (this is the benchmark network for all the further experiments)
- Activation function: Tanh
- Key takeaway:
  - Loss = MSE_{Loss} + PDE_{Loss} (residual loss associated with the PDE)
  - Optimisation is based on this modified physics-informed loss function
- $N_u = 100$; $N_f = 10000$; Num of epochs = 50000
- Optimiser: L-BFGS optimiser

$$MSE = MSE_u + MSE_f,$$

where

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

and

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

Physics-Informed Neural Networks: By M Raissi

Results:

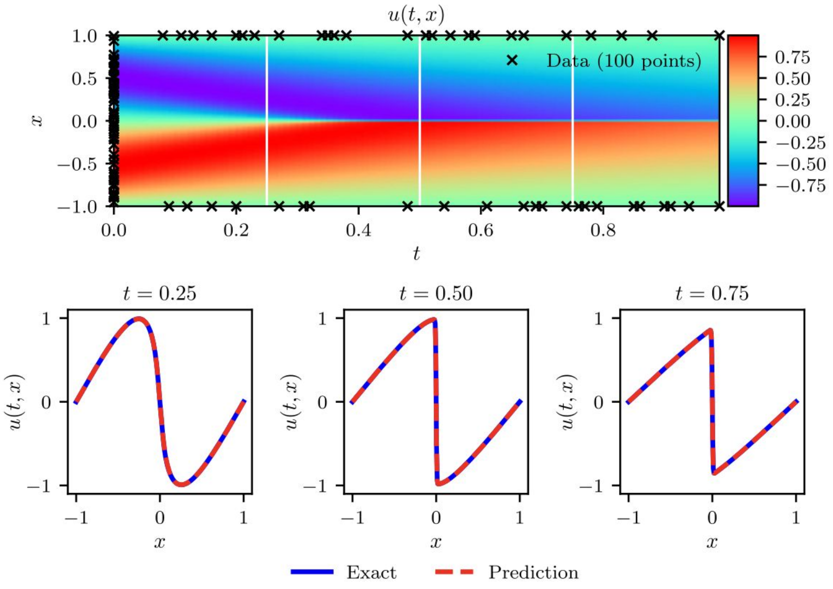


Table A.1

Burgers' equation: Relative $\mathbb{L}_2$ error between the predicted and the exact solution $u(t,x)$ for different number of initial and boundary training data N_u, and different number of collocation points N_f. Here, the network architecture is fixed to 9 layers with 20 neurons per hidden layer.

N_u \ N_f	2000	4000	6000	7000	8000	10000
20	2.9e−01	4.4e−01	8.9e−01	1.2e+00	9.9e−02	4.2e−02
40	6.5e−02	1.1e−02	5.0e−01	9.6e−03	4.6e−01	7.5e−02
60	3.6e−01	1.2e−02	1.7e−01	5.9e−03	1.9e−03	8.2e−03
80	5.5e−03	1.0e−03	3.2e−03	7.8e−03	4.9e−02	4.5e−03
100	6.6e−02	2.7e−01	7.2e−03	6.8e−04	2.2e−03	6.7e−04
200	1.5e−01	2.3e−03	8.2e−04	8.9e−04	6.1e−04	4.9e−04

Table A.2

Burgers' equation: Relative $\mathbb{L}_2$ error between the predicted and the exact solution $u(t,x)$ for different number of hidden layers and different number of neurons per layer. Here, the total number of training and collocation points is fixed to $N_u = 100$ and $N_f = 10{,}000$, respectively.

Layers \ Neurons	10	20	40
2	7.4e−02	5.3e−02	1.0e−01
4	3.0e−03	9.4e−04	6.4e−04
6	9.6e−03	1.3e−03	6.1e−04
8	2.5e−03	9.6e−04	5.6e−04

Training PINNs - Variations

Other variations of Uniform sampling:

The following variations have been tried out for the training of PINN using uniform sampling:

1. Using ReLU activation function
2. Using ADAM optimiser
3. Regularisation using Dropout (p=0.05)
4. Optimisation using Genetic Algorithms

It has been observed that the above methods lead to early stagnation of training; thus giving a very poor performance.

This could be a potential reason why these methods are not being used in any literature available so far

Adaptive Self-Supervision Algorithms for PINN

Motivation:

- For complex PDEs with large number of inputs, the number of collocation points required increases drastically
- Therefore smarter algorithms have to be designed to sample collocation points which can give same performance with less number of points and in fewer training epochs
- In this paper, sampling collocation points based on a proxy for the probability at the grid points is done. Proxy methods: (1) gradient value (2) residual value (3) resample uniformly when optim stalls
- Algorithms have been tested on inferring Burgers 1D equation; same bench network

Self supervision adaptive training - cosine annealing discussed in paper

Adaptive Self-Supervision Algorithms for PINN

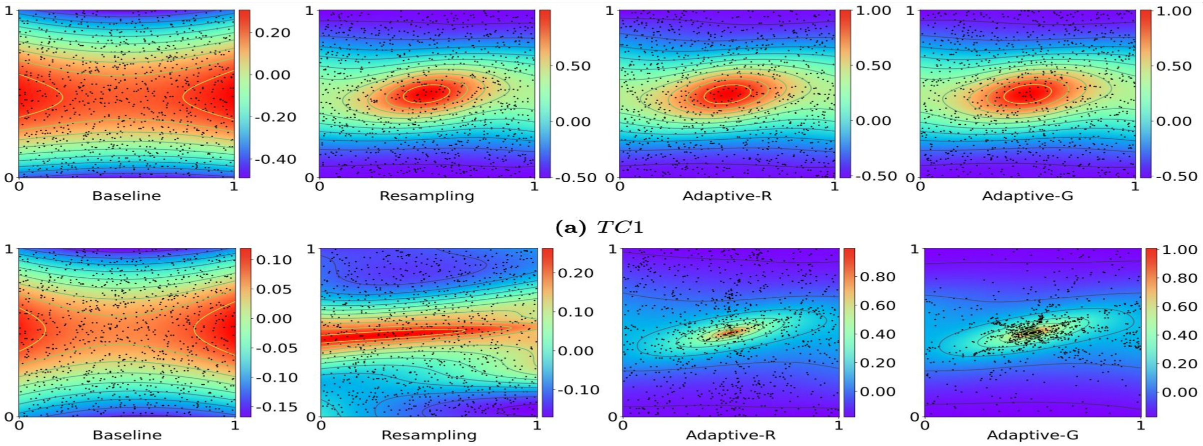
Algorithm 1 Adaptive Sampling for Self-supervision in PINNs

Require: Loss $\mathcal{L}$, NN model, number of collocation points n_c, PDE regularization $\lambda_{\mathcal{F}}$, T, s_w, momentum γ, max epochs $i_{\max}$

$i \leftarrow 0$
while $i \leq i_{\max}$ **do**
  Compute proxy function as the loss gradient (ADAPTIVE-G) or PDE residual (ADAPTIVE-R)
  Current proxy $\mathcal{P}_i \leftarrow \mathcal{P} + \gamma \mathcal{P}_{i-1}$
  $T_c \leftarrow i \bmod T$ ← **Scheduler**
  $\eta \leftarrow$ cosine-schedule(T_c, T) ← **Cosine Annealing = 1/2[1 + cos(Tc/T)]**
  if i mod e is true **then**
    Sample ηn_c points $\boldsymbol{x}_u$ uniformly
    Sample $(1-\eta) n_c$ points $\boldsymbol{x}_a$ using proxy function $\mathcal{P}_i$ **Sampling**
  end if
  $\boldsymbol{x}_c \leftarrow \boldsymbol{x}_u \cup \boldsymbol{x}_a$
  Input $\boldsymbol{x} \leftarrow \boldsymbol{x}_b \cup \boldsymbol{x}_c$ where $\boldsymbol{x}_b$ are boundary points
  $u \leftarrow NN(\boldsymbol{x}; \theta)$
  $\mathcal{L} \leftarrow \mathcal{L}_{\mathcal{B}} + \lambda_{\mathcal{F}} \mathcal{L}_{\mathcal{F}}$
  $\theta \leftarrow$ optimizer-update$(\theta, \mathcal{L})$
  if stopping-criterion is true **then**
    reset cosine scheduler $T_c \leftarrow 0$
  end if
  $i \leftarrow i + 1$
end while

Adaptive Self-Supervision Algorithms for PINN

The algorithms have been implemented for 2D Poisson's equation and 2D diffusion advection equation



(a) TC1: smooth source function $\boldsymbol{\sigma}_f = 0.1$

(b) TC2: sharp source function $\boldsymbol{\sigma}_f = 0.01$

The algorithms have not been tested on 1D Burgers equation in this paper

Test-case	Method	$n_c = 500$	$n_c = 1000$	$n_c = 2000$	$n_c = 4000$	$n_c = 8000$
	Baseline	4.41E-1	5.34E-1	2.73E-2	4.70E-2	5.39E-1
TC1	RESAMPLING	2.53E-2	**1.61E-2**	**2.14E-2**	**2.10E-2**	1.98E-2
	ADAPTIVE-G	**1.94E-2**	1.80E-2	2.17E-2	2.59E-2	**1.79E-2**
	Baseline	7.64E-1	7.09E-1	7.34E-1	6.93E-1	5.44E-1
TC2	RESAMPLING	3.85E-1	4.83E-1	6.74E-2	4.68E-2	5.85E-2
	ADAPTIVE-G	**4.86E-2**	**4.08E-2**	**4.43E-2**	**3.55E-2**	**3.91E-2**

Adaptive Sampling and its Variations (Our Implementations)

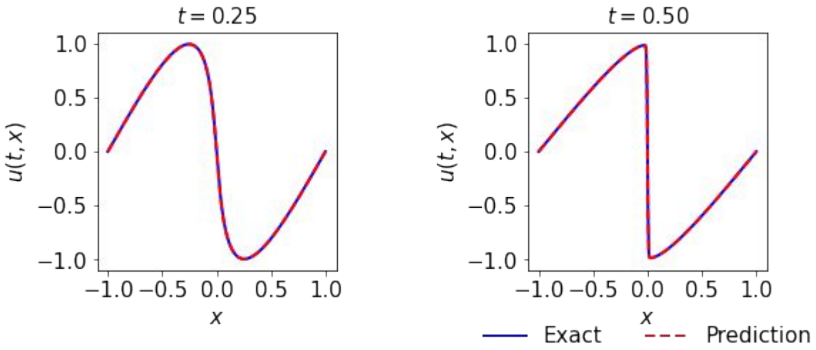
Uniformly Sampled Collocation Points

8 layers: 20 neurons each
No of epochs: 10000
Loss = MSE_{loss} + 1*PDE_{Loss}

Xavier Initialisation of weights
Activation function: Tanh

Training error: 6.93e-6

Test error: 6.86e-3



Adaptive Sampling and its Variations (Our Implementations)

1. Cosine Annealing Proposed in the earlier paper for 1D Burger Equation:
2. Gibbs Probability distribution with cosine annealing:
   - $$P_i = \frac{exp(r_i \ / \ T)}{\sum exp(r_i \ / \ T)}$$
3. Uniform Resampling:
   - In order to analyse the problem of exploding gradients, we have trained using N_f collocation points which are sampled uniformly after ‘e’ epochs
4. But the model did not train due to exploding gradients beyond a certain number of epochs
5. The observation that the same is happening in uniform re-sampling suggests that the problem does not lie in sampling according our probabilistic model
6. This behaviour was not observed when the number of collocation points were high (~6000 - 10000)

Adaptive Sampling and its Variations (Our Implementations)

What was actually happening?

<u>Observations</u>:

- Max probability sometimes is extremely high (250x times uniform probability)
- The first step after this sampling is happening
- The forward inference in the next step results in extremely high MSE_loss and PDE_loss
- Backward step on this huge loss results in exploding gradients
- PDE_Loss is extremely high compared to MSE_Loss

```
iter: 4330 startin loss_func
iter: 4330 computing net_u
iter: 4330 net_u max = 28.059926986694336
iter: 4330 computing net_f
iter: 4330 net_f max = 3981566.5
computing u-loss
u-loss = 158.1429901123047
computing f-loss
f-loss = 1.7704674880030507e+19
iter: 4330 taking gradient step: loss.backward
iter 4331, Loss: 1.77047e+19, Loss_u: 1.58143e+02,
Loss_f: 1.77047e+19
iter: 4331 returning loss
-------------------------------------
iter: 4331 startin loss_func
iter: 4331 computing net_u
iter: 4331 net_u max = nan
iter: 4331 computing net_f
iter: 4331 net_f max = nan
computing u-loss
u-loss = nan
computing f-loss
f-loss = nan
iter: 4331 taking gradient step: loss.backward
iter 4332, Loss: nan, Loss_u: nan, Loss_f: nan
iter: 4332 returning loss
```

Adaptive Sampling and its Variations (Our Implementations)

<u>Possible Explanation</u>:

- After training for a few epochs, the weights of the model get adjusted according to the collocation points sampled so far. When new points are being sampled based on their residual/gradient values, a few of the points with extremely high residual values and being sampled.
- The gradient step taken according to this loss results in a large change in weights of the model. Therefore, when the model is fed forward for the next iteration, the total loss increases largely due to the exploding of PDE loss
- Here, the contribution of PDE loss to the total is in same proportion as MSE loss

<u>Possible Solution</u>:

- Scale down the contribution of PDE_Loss to the total loss
- Loss = $MSE_{Loss} + \boldsymbol{\lambda}\ PDE_{Loss}$
- But, the drawback with this scaling is it requires more optimization steps for training

Adaptive Sampling and its Variations (Our Implementations)

Uniformly Sampled (no re-sampling)

Loss = MSE_{Loss} + 0.01*PDE_{Loss}

Nf = 2000, Nu = 100

Training Error:
iter 3473, Loss: 4.70353e-06, Loss_u: **5.34757e-07**, Loss_f: 4.16877e-04

Test Error:
Error u: **2.074267e-01**

Uniform Re-Sampling

Loss = MSE_{Loss} + 0.01*PDE_{Loss}

Nf = 2000, Nu = 100
e = 100 (after every 100 epochs resample)

Training Error:
iter 7771, Loss: 1.23169e-06, Loss_u: **1.41283e-07**, Loss_f: 1.09041e-04

Test Error:
Error u: **1.682376e-02**

Adaptive Sampling and its Variations (Our Implementations)

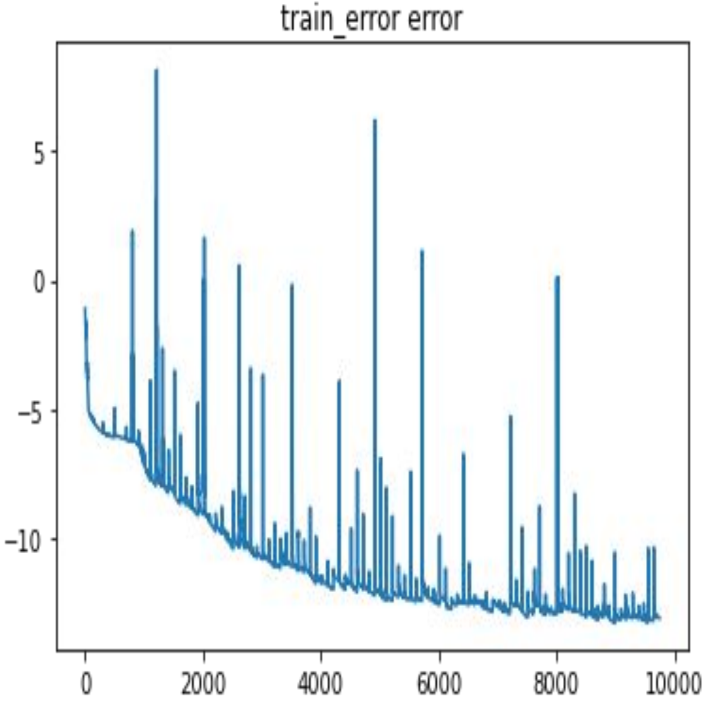
Gibbs Annealing (with new loss function)

$N_f = 2000$
$N_u = 100$

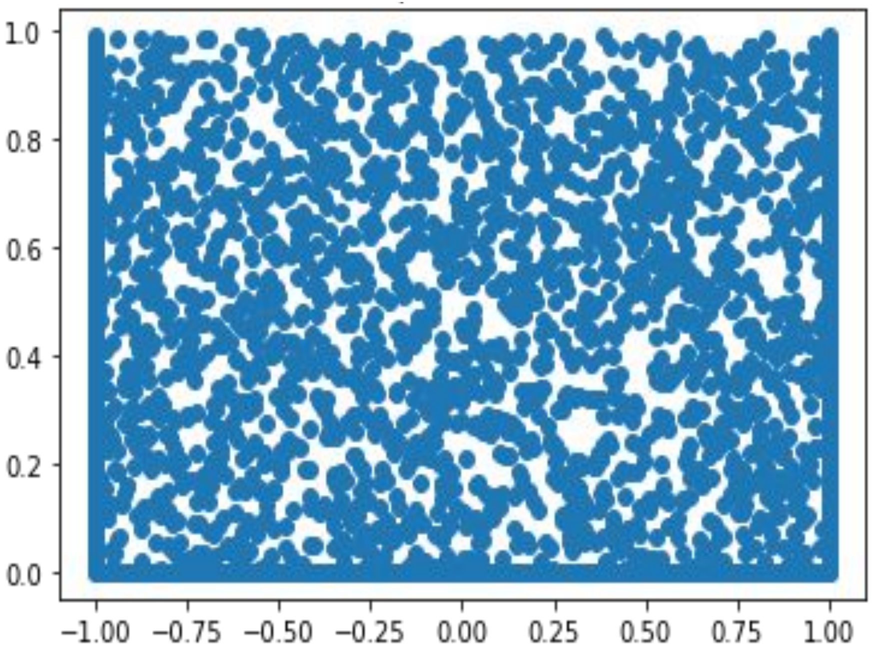
e = 100 epochs
Temperature: 20.0 -> 0.9698
rate of decrease = 10% for every 100 epochs)

iter 9752, Loss: 2.16710e-06, Loss_u: **1.37280e-07**, Loss_f: 2.02982e-04

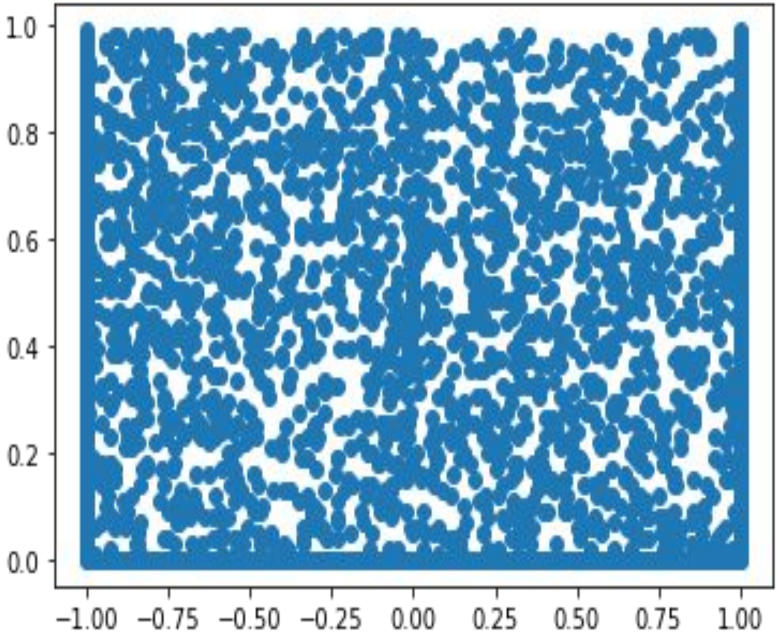
Test Error:
Error u: **6.744674e-03**



Adaptive Sampling and its Variations (Our Implementations)



N_f = 2000 Uniform Resampling



N_f = 2000 Gibbs Annealing Sampling

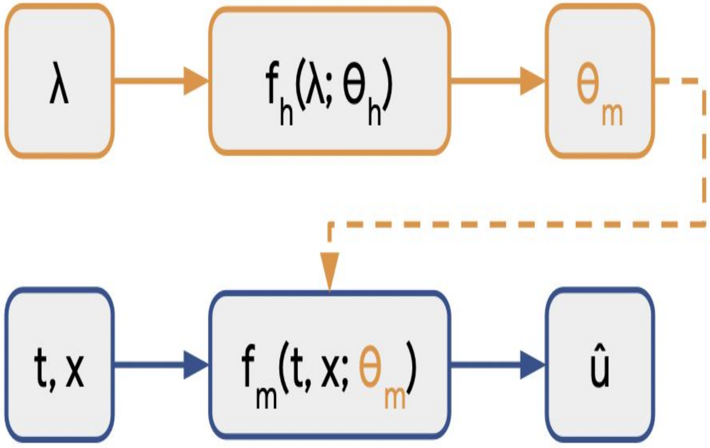
Conclusions

Sampling Method	**Accuracy**
Uniform Sampling (non resampling)	**2.074267e-01**
Uniform Resampling	**1.682376e-02**
Cosine annealing ($MSE = MSE_{Loss} + 0.001*PDE_{Loss}$)	**2.86122e-02**
Gibbs Annealing (T = 20 to 1, -10%)	**6.744674e-03**

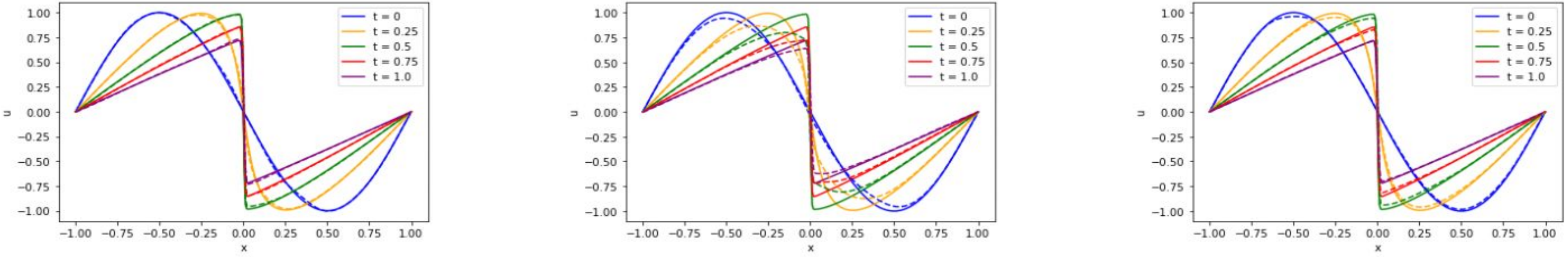
HyperPINN: Learning Parameterized Differential Equations with Physics-informed hypernetworks

- A new network for generating weights based on parameter value is augmented with PINN
- $\lambda = [0.001, 0.1]$; $x = [-1,1]$; $t = [0,1]$
- $N_u = 100$
- No information on N_f and number of epochs
- The results obtained are promising, even for

  $\nu = 0.001$



HyperPINN: Learning Parameterized Differential Equations with Physics-informed hypernetworks



(a) HyperPINN (b) Small PINN baseline (c) Large PINN baseline

Table 1: Comparison of HyperPINN and baselines on the parameterized Burgers' PDE.

Model	Mean squared error	Model size	Evaluation time
Small PINN	$3.0 \cdot 10^{-4}$	401 parameters	$92\mu s$
Large PINN	$2.3 \cdot 10^{-5}$	9665 parameters	$158\mu s$
HyperPINN	$1.9 \cdot 10^{-5}$	Main: 393 parameters Hyper: 9385 parameters	Main: $86\mu s$ Hyper: $158\mu s$

Deep Theory of Functional Connections

- We convert the constrained problem into an unconstrained problem

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], t \in [0, 1],$$
$$u(x, 0) = -\sin(\pi x),$$
$$u(-1, t) = u(1, t) = 0,$$

$$f(x_1, g(x_1)) = g(x_1) + \sum_{j=1}^{2} \eta_j s_j(x_1),$$

$$\mathcal{L}(x_1) = \left[m\frac{\mathrm{d}^2 f(x_1, g(x_1))}{\mathrm{d}x_1^2} + k f(x_1) \right]^2$$

- $g(x_1)$ is approximated with a neural network in Deep TFC
- η_j's are obtained using the constraints (in this case- BC + IC)
- S_j are the support functions (typically monomials are chosen)

A Recent Paper on Sampling Methods

- The paper discusses various methods of sampling collocation points
- Three new residual based adaptive sampling techniques have been proposed
    - Uniformly Distributed Non-Resampling Points
        - Grid, Random, LHS, Halton, Hammersley, Sobol
    - Uniform Points with Re-sampling
    - Non-uniform adaptive sampling
        - RAR-G (Residual Based Adaptive Refinement with Greed)
        - RAD - Residual based adaptive distribution
        - RAR-D (Residual Based Adaptive Refinement with distribution)

A Recent Paper on Sampling Methods

Algorithm 1: RAR-G [3].

Sample the initial residual points $\mathcal{T}$ using one of the methods in Section 2.2.1;
Train the PINN for a certain number of iterations;
repeat
    Sample a set of dense points $\mathcal{S}_0$ using one of the methods in Section 2.2.1;
    Compute the PDE residuals for the points in $\mathcal{S}_0$;
    $\mathcal{S} \leftarrow m$ points with the largest residuals in $\mathcal{S}_0$;
    $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{S}$;
    Train the PINN for a certain number of iterations;
until *the total number of iterations or the total number of residual points reaches the limit*;

Algorithm 2: RAD.

Sample the initial residual points $\mathcal{T}$ using one of the methods in Section 2.2.1;
Train the PINN for a certain number of iterations;
repeat
    $\mathcal{T} \leftarrow$ A new set of points randomly sampled according to the PDF of Eq. (2);
    Train the PINN for a certain number of iterations;
until *the total number of iterations reaches the limit*;

A Recent Paper on Sampling Methods

Algorithm 3: RAR-D.

Sample the initial residual points $\mathcal{T}$ using one of the methods in Section 2.2.1;
Train the PINN for a certain number of iterations;
repeat
    $\mathcal{S} \leftarrow m$ points randomly sampled according to the PDF of Eq. (2);
    $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{S}$;
    Train the PINN for a certain number of iterations;
until *the total number of iterations or the total number of residual points reaches the limit*;

Table 1
The hyperparameters used for each numerical experiment.

Problems	Depth	Width	Optimizer
Section 3.2 Diffusion equation	4	32	Adam
Section 3.3 Burgers' equation	4	64	Adam + L-BFGS
Section 3.4 Allen–Cahn equation	4	64	Adam + L-BFGS
Section 3.5 Wave equation	6	100	Adam + L-BFGS
Section 3.6 Diffusion–reaction equation (inverse)	4	20	Adam
Section 3.7 Korteweg–de Vries equation (inverse)	4	100	Adam

The learning rate of Adam optimizer is chosen as 0.001.

A Recent Paper on Sampling Methods

Table 2
L^2 relative error of the PINN solution for the forward problems.

	Diffusion	Burgers'	Allen–Cahn	Wave
No. of residual points	30	2000	1000	2000
Grid	$0.66 \pm 0.06\%$	$13.7 \pm 2.37\%$	$93.4 \pm 6.98\%$	$81.3 \pm 13.7\%$
Random	$0.74 \pm 0.17\%$	$13.3 \pm 8.35\%$	$22.2 \pm 16.9\%$	$68.4 \pm 20.1\%$
LHS	$0.48 \pm 0.24\%$	$13.5 \pm 9.05\%$	$26.6 \pm 15.8\%$	$75.9 \pm 33.1\%$
Halton	$0.24 \pm 0.17\%$	$4.51 \pm 3.93\%$	$0.29 \pm 0.14\%$	$60.2 \pm 10.0\%$
Hammersley	$0.17 \pm 0.07\%$	$3.02 \pm 2.98\%$	**$0.14 \pm 0.14\%$**	$58.9 \pm 8.52\%$
Sobol	$0.19 \pm 0.07\%$	$3.38 \pm 3.21\%$	$0.35 \pm 0.24\%$	$57.5 \pm 14.7\%$
Random-R	**$0.12 \pm 0.06\%$**	$1.69 \pm 1.67\%$	$0.55 \pm 0.34\%$	**$0.72 \pm 0.90\%$**
RAR-G [3]	$0.20 \pm 0.07\%$	**$0.12 \pm 0.04\%$**	$0.53 \pm 0.19\%$	$0.81 \pm 0.11\%$
RAD	**$\underline{0.11 \pm 0.07}\%$**	**$\underline{0.02 \pm 0.00}\%$**	**$\underline{0.08 \pm 0.06}\%$**	**$\underline{0.09 \pm 0.04}\%$**
RAR-D	**$0.14 \pm 0.11\%$**	**$0.03 \pm 0.01\%$**	**$0.09 \pm 0.03\%$**	**$0.29 \pm 0.04\%$**

Future Works

- Implement TFC for 1D Burgers equation and understand the effect of support functions and validate adaptive methods on deep TFC
- Using HyperPINN for generating weights for 1D Burgers equation
  - Try to generate weights for each pair of hidden layers using a hypernetwork and test its performance w.r.t using only one hypernetwork
- Improve Gibbs Annealing, possible ideas:
  - Keep increasing the lambda as training happens
- Use the developed algorithms for solving HJB equation in control problem

Thank You