B.Tech Project Report: Phase 1

Physics-Informed Neural Networks

By

N V Sai Gangadhar

190100080

Guide:
Dr. Asim Tewari
Mechanical, IIT Bombay

Co-Guide:
Dr. Mayank Baranwal
SysCon, IIT Bombay

Mechanical Engineering

Indian Institute of Technology, Bombay

November 24, 2022

INTRODUCTION

Physical Systems are all about modelling and solving the model to achieve the desired output. Although modelling is a task that requires understanding the physical system, solving the model in itself is a complex task which requires a high computational power. Most of these physical systems are modelled in partial-differential equations and these equations along with the boundary and initial conditions encompass the entire system. One way to solve these PDEs is to use machine learning algorithms. But, 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, and therefore the vast majority of SOTA ML techniques are lacking robustness and fail to provide any guarantee on convergence.

Physics Informed Neural Networks, are the neural networks in which the physics governing the systems is encoded into the total loss along with the supervised loss. PINNs have shown to be effective tools for solving both forward and inverse problems of partial differential equations (PDEs). PINNs embed the PDEs into the loss of the neural network using automatic differentiation, and this PDE loss is evaluated at a set of scattered spatio-temporal points (called residual points). 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:
$u_t + N[u; \lambda] = 0$, $x \in \Omega$, $t \in [0, T]$. where u(t,x) denotes the latent (hidden) solution that we are interested in solving. $N[u; \lambda]$ is a non-linear operator parametrized by λ. N_u = Number of supervised training samples; N_f = Number of Collocation Points A little background on 1D Burger's Equation. All the experiments done on the 1D Burgers equation consider the following details:

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

LITERATURE REVIEW

PINNs (benchmark network for 1D burgers equation):

This paper by M. Raissi is the first proper implementation of Physics-Informed Neural Networks. The network is optimized and tested for solving the 1D burgers equation. We define $f(t,x)$ to be given by the left-hand side as f := ut + N [u], and proceed by approximating u(t, x) by a deep neural network. This assumption along with equation (3) results in a physics-informed neural network f (t, x). This network can be derived by applying the chain rule for differentiating compositions of functions using automatic differentiation.

$$MSE = MSE_u + MSE_f,$$

where

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

and

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

A deep neural network of 8 layers and 20 neurons each is reported to be the most optimal for inferring 1D burgers equation. In this report, N_u = 100 (supervised learning) and N_f = 10000 is reported. A experiment varying the values of N_u and N_f is also reported along with variations in the number of layers and neurons per layer. Key takeaways from this paper is that physics information can be

embodied into the loss for inferring the function determined by PDE. The activation function used is Tanh.

Results reported in the paper are as follows:

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.

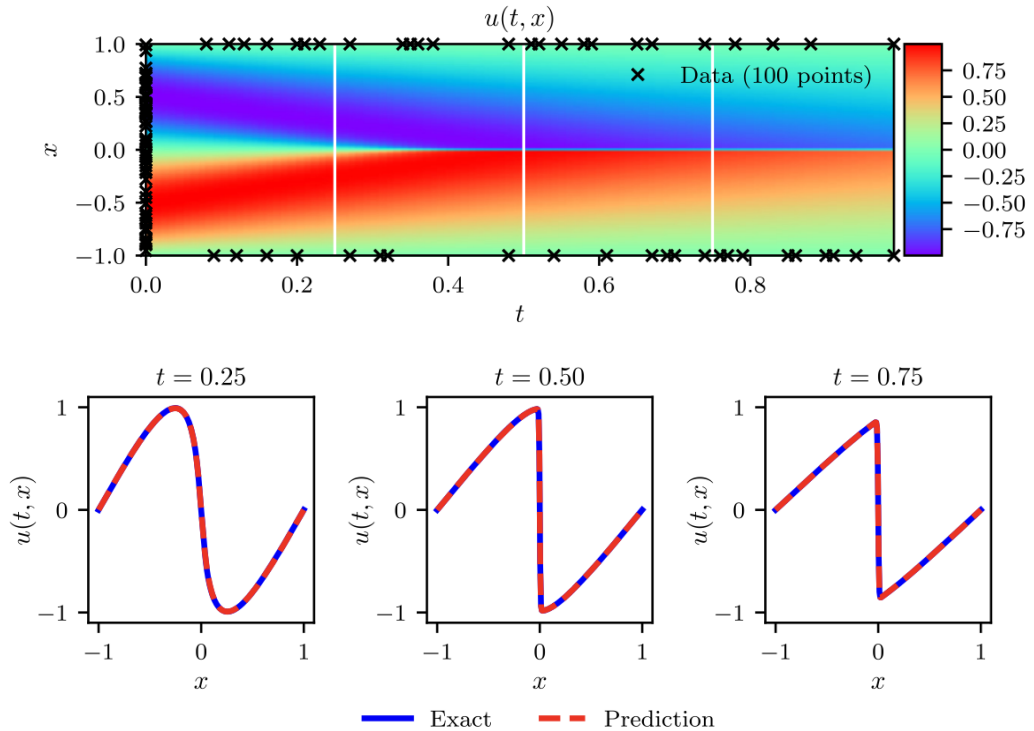
N_u \ N_f	2000	4000	6000	7000	8000	10000
20	2.9e−01	4.4e−01	8.9e−01	1.2e+00	9.9e−02	4.2e−02
40	6.5e−02	1.1e−02	5.0e−01	9.6e−03	4.6e−01	7.5e−02
60	3.6e−01	1.2e−02	1.7e−01	5.9e−03	1.9e−03	8.2e−03
80	5.5e−03	1.0e−03	3.2e−03	7.8e−03	4.9e−02	4.5e−03
100	6.6e−02	2.7e−01	7.2e−03	6.8e−04	2.2e−03	6.7e−04
200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

In Table A.1 we report the resulting relative L2 error for a different number of initial and boundary training data Nu and a different number of collocation points N_f while keeping the 9-layer network architecture fixed. The general trend shows increased prediction accuracy as the total number of training data Nu is increased, given a sufficient number of collocation points N_f. Table A.2 shows the resulting relative L2 for different numbers of hidden layers, and a different number of neurons per layer, while the total number of training and collocation points is kept fixed to Nu = 100 and N f = 10,000.



The above figure summarizes our results for the data-driven solution of the Burgers equation. Specifically, given a set of Nu = 100 randomly distributed initial and boundary data, we learn the latent solution u(t, x) by training all 3021 parameters of a 9-layer deep neural network using the mean squared error.

Adaptive Self-Supervision Algorithms for PINN:

Training PINNs this way can be very difficult, and it is often difficult to solve the optimization problem [6, 11, 25]. This could be partly because the self-supervision term typically contains complex terms such as (higher-order) derivatives of spatial functions and other nonlinearities that cause the loss term to become ill-conditioned. One overlooked, but very important, parameter of the training process is the way that the self-supervision is performed in PINNs, and in particular which data points in the domain are used for enforcing the physical constraints (commonly referred to as collocation points in numerical analysis.

In this paper, a novel approach of utilising the loss information at various points is used to sample collocation points strategically. The paper proposes to use the following methods as a proxy for obtaining probabilities associated with sampling a collocation point from the domain:

(1) Residual as proxy (2) Gradient as proxy (3) Resampling when optimizer stalls

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

Test-case	Method	$n_c = 500$	$n_c = 1000$	$n_c = 2000$	$n_c = 4000$	$n_c = 8000$
TC1	Baseline	4.41E-1	5.34E-1	2.73E-2	4.70E-2	5.39E-1
	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**
TC2	Baseline	7.64E-1	7.09E-1	7.34E-1	6.93E-1	5.44E-1
	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**

For 2D Poisson's TC1: smooth source function $\sigma_f = 0.1$ and TC2: sharp source function $\sigma_f = 0.01$

Key takeaway from this paper is the idea of using residuals and gradients as a proxy for probabilities for sampling collocation points. Another key takeaway is the use of cosine annealing. Here, as the number of epochs increases the fraction of uniformly sampled points decreases, and points sampled according to probability distribution increase. This can be used in further works on adaptive sampling.

EXPERIMENTS

Variations of Simple PINN model:

Most of the literature available consists of the following for hyperparameter selection. Activation function is always Tanh**,** optimiser used is L-BFGS in most of the cases. Also it can be observed that in most of the cases the model was overfitting a lot. The difference between the order of training error and testing error was about 100-1000 times. Therefore, with a motivation to address these, the following experiments have been conducted on benchmark PINN model for 1D Burgers equation.

(1) ReLU activation function (2) ADAM Optimiser (3) Regularisation with dropout (4) GA

But, surprisingly the model did not train with these hyperparameter values. The model was stalling after a few epochs thus producing a very poor performance. We believe that this is greatly due to the highly complex loss function introduced because of PDE loss, and therefore is not being used anywhere in the literature.

Adaptive Sampling Variations:

The Adaptive sampling methods discussed in the earlier paper was tested only on 2D poissons and 2D Diffusion Advection equation. We implemented the cosine-annealing algorithm for adaptive sampling on 1D Burgers equation. Note the loss function that has been used till this point is given by Loss = MSE_{Loss} + 1*PDE_{Loss} (i.e. the regularisation factor for PDE loss is equal to 1). It was observed that the gradients were exploding beyond a certain number of training epochs.

Variation1:

With a motivation to improve the probability distribution, we implemented Gibbs Probability distribution with cosine annealing. Gibbs Probability distribution is given by: $P_i = \frac{exp(r_i/T)}{\Sigma exp(r_i/T)}$

Variation2:

The earlier uniform sampling method involved sampling all the N_f collocation points uniformly only once and using the same set of points for training over all the epochs. We implemented a Uniform Resampling method where N_f points are resampled uniformly after every ‘e’ epochs.

OBSERVATIONS AND CONCLUSIONS

The model did not train due to exploding gradients beyond a certain number of epochs for all the three variations that we have experimented with. The observation that the same is happening in uniform re-sampling suggests that the problem does not lie in sampling according to our probabilistic model. This behavior was not observed when the number of collocation points was high (~6000 - 10000).

A possible explanation for this behavior is that the newly sampled points cause a very high PDE loss which leads to a large value in total loss. More points will be sampled where there is a high residual value (PDE loss) and therefore after a large number of epoch training (towards the end of training), there is a spike in the residual loss leading to gradients exploding in the next step. 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 the 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.

A sample output snippet when training is performed on above methods:

```
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
```

MODIFIED APPROACH

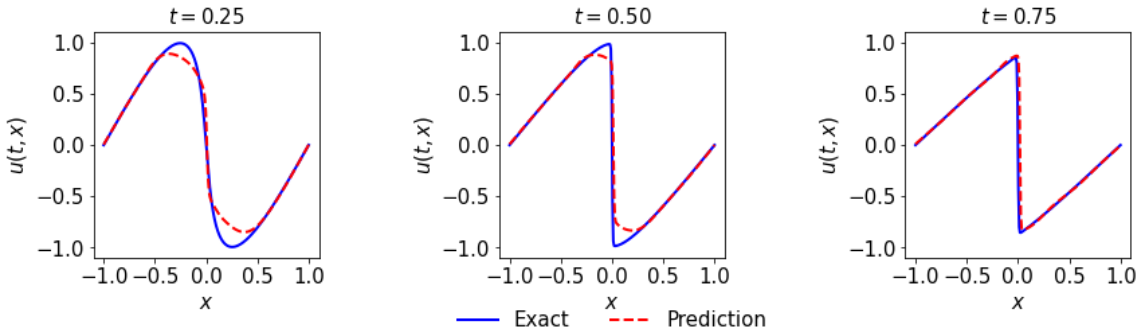
One of the approaches to deal with these exploding gradients is to introduce a regularisation term which will scale down the effect of PDE loss compared to MSE loss. But, the drawback with this scaling is it requires more optimization steps for training. Loss = MSE_{Loss} + $\boldsymbol{\lambda}$ PDE_{Loss} where $\boldsymbol{\lambda}$ is typically between 0.01 and 0.001

RESULTS

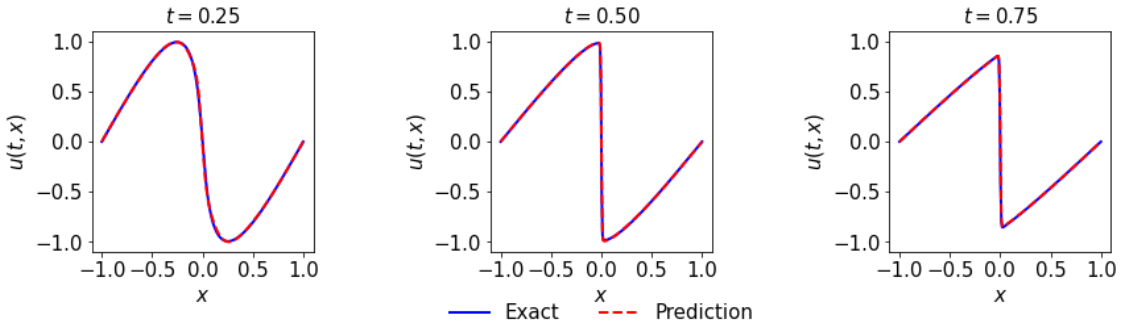
An exhaustic set of results for all the experimentation done with uniformly sampled PINN and adaptive sampling are presented here.

Simple Implmentation of PINN for 1D Burgers Equation:

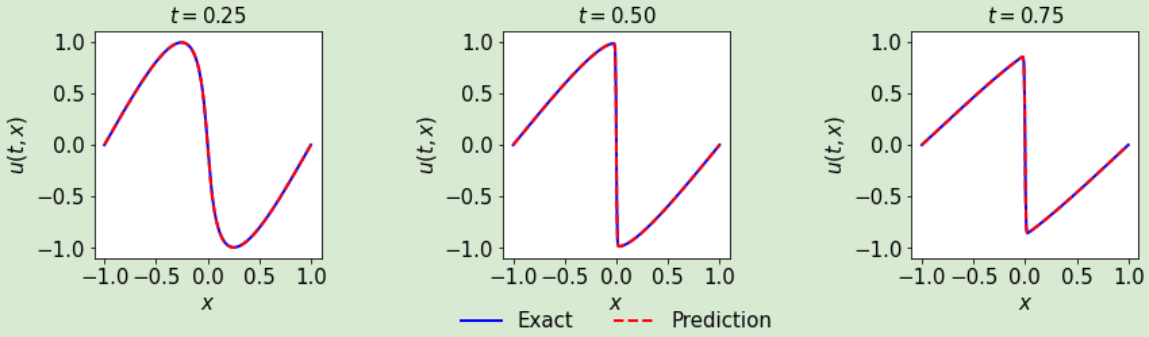
8 layers : [200, 200, 200, 200, 200, 200, 200, 200]
No of epochs: 700
Training error: 0.006857
Test error: 0.188



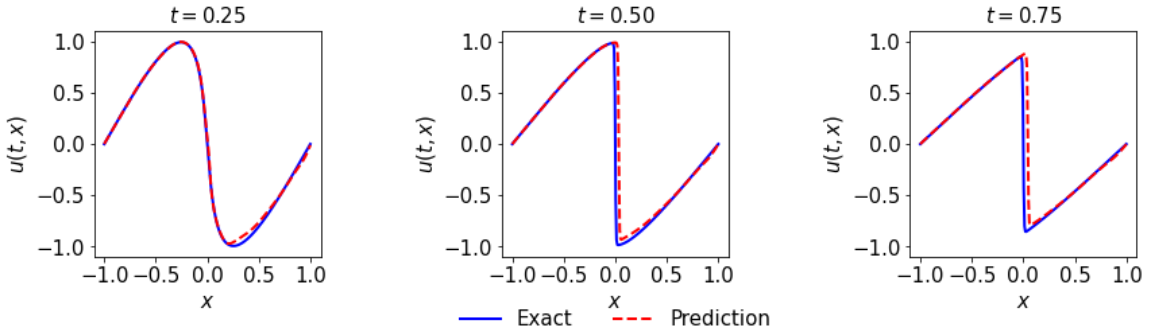
8 layers : [200, 200, 200, 200, 200, 200, 200, 200]
No of epochs: 10000 (training stopped after 3800 epochs, error saturated)
Training error: 1.32e-4
Test error: 0.01622



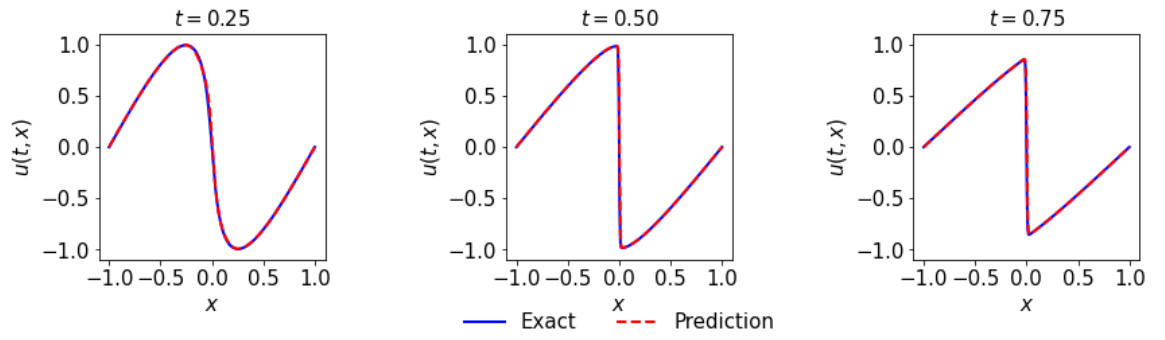
8 layers: 20 each
No of epochs: 10000
Training error: 6.93e-6
Test error: 6.86e-3



8 layers: 20 each
No of epochs: 10000
Normalisation : (x - mu / sigma)
Training error: 1.339e-03
Test error: 2.956e-01



4 layers: 200 each
No of epochs: 10000
Training error: 1.05e-4
Test error: 4.09e-2
?.



Adaptive Sampling with regularised physics-informed loss:

Using proportionality as probabilities:
Loss = L_u + 0.001*L_f
Gamma = 0.1
Num of epochs = 10000
Num of collocation points = 2000

```
iter 9891, Loss: 5.79062e-06, Loss_u: 7.19358e-07, Loss_f:
5.07127e-03
Test error: Error u: 2.861200e-02
```

Uniform non-adaptive sampling
Loss = L_u + 0.001*L_f
Num of epochs = 10000
Num of collocation points = 2000

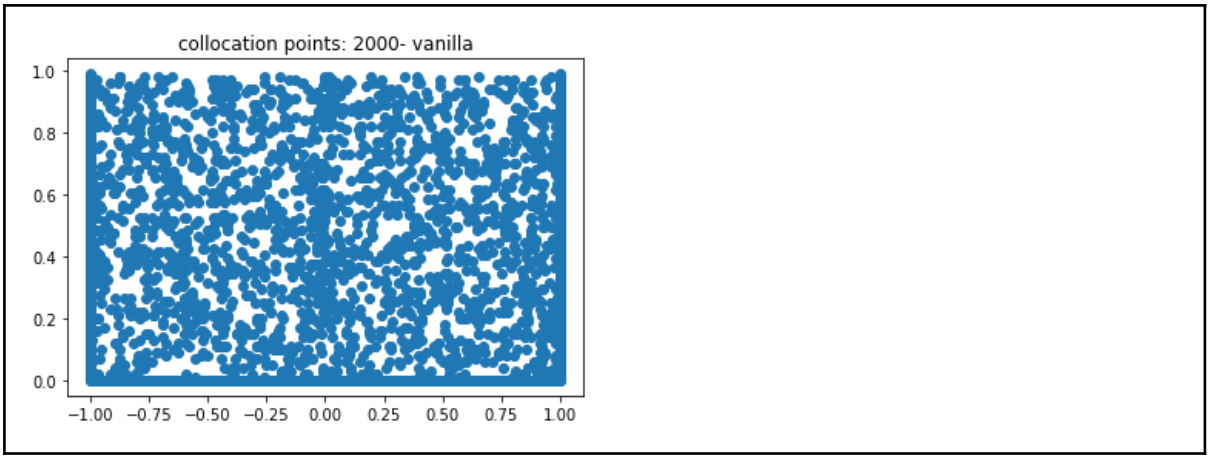
```
iter 2476, Loss: 3.14011e-04, Loss_u: 1.25729e-05, Loss_f:
3.01438e-01
Error u: 3.479325e-01
```

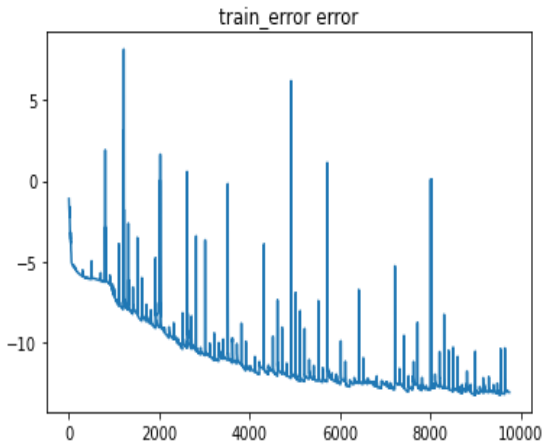
Using Gibbs sampling: T = 20 - 1.0 (with a decrease of 10% after 100 epochs)
Loss = L_u + 0.01*L_f
Gamma = 0.0
Num of epochs = 10000
Num of collocation points = 2000

```
iter 9824, Loss: 4.02361e-06, Loss_u: 3.68469e-07, Loss_f:
3.65514e-04
Error u: 8.753277e-03
```

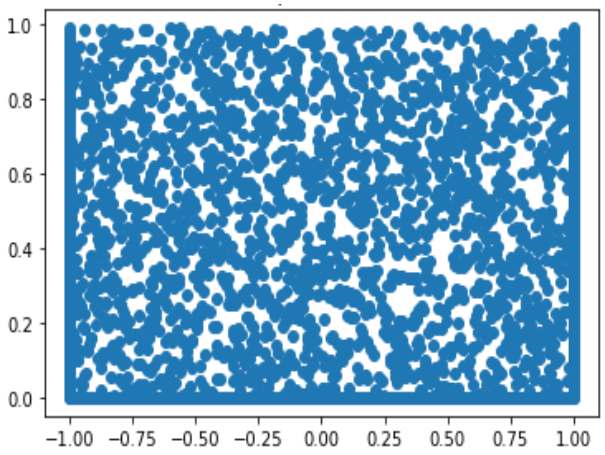
Using Gibbs sampling: T = 20 - 0.5 (with a decrease of 10% after 100 epochs)
Loss = L_u + 0.01*L_f
Gamma = 0.0
Num of epochs = 10000
Num of collocation points = 2000

```
iter 9825, Loss: 4.78648e-06, Loss_u: 3.28305e-07, Loss_f:
4.45818e-04
Error u: 8.919903e-03
```





Training error for adaptive sampling



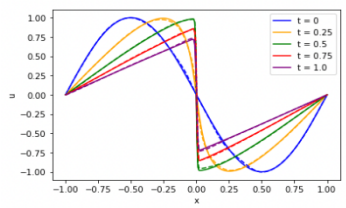
uniformly resampled

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**

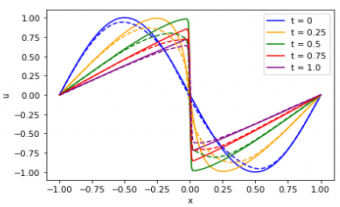
FUTURE WORKS

HyperPINN: Learning Parameterized Differential Equations with Physics-informed hypernetworks

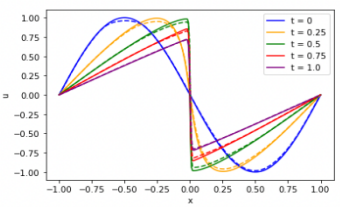
The paper proposed by Google discusses the use of Hypernetworks for PINNs to generalise the model over the range of parameter inputs. This model can also be extended to generalising over boundary conditions and initial conditions



(a) HyperPINN



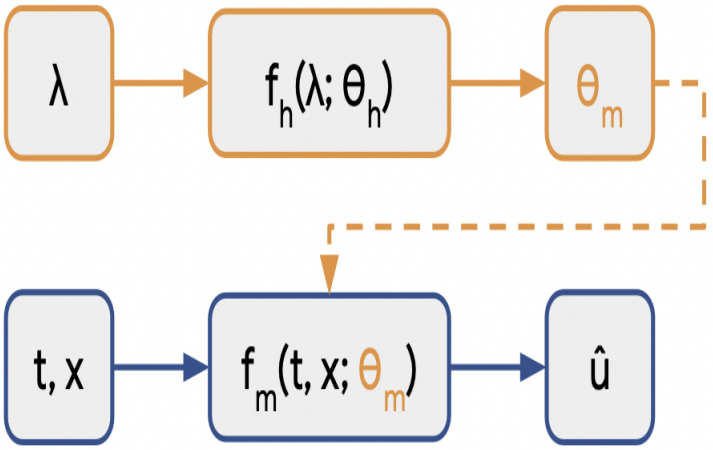
(b) Small PINN baseline



(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: $158\mu s$



The training is done over $\lambda = [0.001, 0.1]$ for $N_u = 100$ points and both the networks are trained using both supervised points and collocation points. For encompassing the information of boundary conditions and initial conditions, we may have to use encode-decoder network for extracting the universal physics from the conditions.

Deep Theory of Functional Connections (TFC):

The Theory of Functional Connections (TFC) is a mathematical framework designed to turn constrained problems into unconstrained problems. This is accomplished through the use of constrained expressions, which are functionals that represent the family of all possible functions that satisfy the problem's constraints.

TFC has two major steps: (1) Embed the boundary conditions of the problem into the constrained expression; (2) solve the now unconstrained optimization problem. The paragraphs that follow will explain these steps in more detail.

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

$$f(x_1, g(x_1)) = g(x_1) + \sum_{j=1}^{2} \eta_j s_j(x_1),$$
$$\mathscr{L}(x_1) = \left[m\frac{\mathrm{d}^2 f(x_1, g(x_1))}{\mathrm{d}x_1^2} + k f(x_1)\right]^2$$

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

Comprehensive Study of Sampling Methods:

This paper discusses about the results obtained from various sampling methods for PINNs and proposes three approaches for adaptive sampling. The paper discusses the three cases of uniformly sampled (non-resampling), uniform re-sampling and non-uniform re-sampling methods. The results are presented below in Table 2. The algorithms proposed are given below.

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*;

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 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 ± 0.06%	13.7 ± 2.37%	93.4 ± 6.98%	81.3 ± 13.7%
Random	0.74 ± 0.17%	13.3 ± 8.35%	22.2 ± 16.9%	68.4 ± 20.1%
LHS	0.48 ± 0.24%	13.5 ± 9.05%	26.6 ± 15.8%	75.9 ± 33.1%
Halton	0.24 ± 0.17%	4.51 ± 3.93%	0.29 ± 0.14%	60.2 ± 10.0%
Hammersley	0.17 ± 0.07%	3.02 ± 2.98%	**0.14 ± 0.14%**	58.9 ± 8.52%
Sobol	0.19 ± 0.07%	3.38 ± 3.21%	0.35 ± 0.24%	57.5 ± 14.7%
Random-R	**0.12 ± 0.06%**	1.69 ± 1.67%	0.55 ± 0.34%	**0.72 ± 0.90%**
RAR-G [3]	0.20 ± 0.07%	**0.12 ± 0.04%**	0.53 ± 0.19%	0.81 ± 0.11%
RAD	**0.11 ± 0.07%**	**0.02 ± 0.00%**	**0.08 ± 0.06%**	**0.09 ± 0.04%**
RAR-D	**0.14 ± 0.11%**	**0.03 ± 0.01%**	**0.09 ± 0.03%**	**0.29 ± 0.04%**

The main objective of this BTP is to study and design algorithms for PINNs for solving optimal control problems, which will be done in phase 2 of my B.Tech Project.

REFERENCES

Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations

Adaptive Self-supervision Algorithms for Physics-informed Neural Networks

HyperPINN: Learning parameterized differential equations with physics-informed hypernetworks

Deep Theory of Functional Connections: A New Method for Estimating the Solutions of Partial Differential Equations

A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural network