First Formal Progress Report

Jose Florido

University of Leeds

School of Computing

January, 2023

Declaration of Academic Integrity

Contents

Abbreviations

ML	Machine Learning	PIML	Physics Informed Machine Learning
NN	Neural Network	PINN	Physics Informed Neural Network
LHS	Latin Hypercube Sampling	TF	TensorFlow
FEM	Finite Element Methods	FDM	Finite Difference Method
CFD	Computational Fluid Dynamics	NSE	Navier Stokes Equations
ODE	Ordinary Differential Equation	PDE	Partial Differential Equation

Chapter 1 : Research Focus

1.1 Introduction / Research Question

Data-driven methods recent popularity have highlighted their many advantages for solving a variety of problems across many applications. Physics-Informed Machine Learning (PIML) methods have been the product of recent developments attempting to combine Machine Learning (ML) methods with the established expertise and pre-existing knowledge of physical problems. By combining the knowledge on the PDE systems that model many fluid dynamics interactions, physics-informed neural networks can be constructed that can solve these PDE systems more accurately than traditional ML methods or without the need for large amounts of data (one of traditional ML's biggest drawbacks). By providing advantages from both ML and traditional numerical methods, PIML has a lot of potential for being a useful method for several applications in science, biomedical and engineering applications. It excels at high-dimensional problems, problems with ill-defined boundary conditions, or in applications that might require on-demand computation of solutions (such as in some medical applications). These types of problems are not efficiently solvable using traditional numerical methods, and regular data-driven methods cannot solve them without large amounts of data, which might be unavailable or require expensive simulations to obtain.

The research question is therefore how can we build on recent developments in PIML to improve the performance of physics informed neural networks without substantial increases to cost. This project will aim to evaluate and understand the current state-of-the-art Physics-Informed Neural Networks (PINNs) being utilised in the literature and explore how they can be improved via incorporating novel techniques or improving their implementation in order to improve the accuracy and computational efficiency of these methods. The desired outcomes of this project would be to develop new algorithms that can be incorporated into PIML models to improve their performance solving fluid dynamics problems.

1.2 Aims

On a general level, this project would aim to improve PINNs performance by modifying their implementation or adding techniques that improve their performance. In particular, would want to improve them on the problems where physics informed machine learning has the most potential to be used in; for example, to solve fluid dynamics PDEs for scenarios with little or no prior data from traditional numerical methods (typically called 'unsupervised learning'). At the moment, one such area with potential was adaptive sampling: looking at intelligently distributing the collocation points in a PINN (the points of the domain at which physics is enforced) in order to bring about benefits to performance. Fully exploring the potential of incorporating this technique into PIML methods will be the main aim of this project for the foreseeable future. However, in order to fulfill the broader research question, this project also aims to identify other areas of PINNs with the most potential for novel improvements to be made and to establish a plan for researching those improvements. As well as looking at forward problems (finding solution using known system of PDEs), could also look at the use of PINNs networks for solving reverse problems, for problems with unknown/incomplete boundary condition information, or with unknown physics equations but available data.

These aims can be summarised in the following way:

- To test whether adaption can bring improvements to performance of PINN network.
- If possible, find extent of this and whether the performance gains are generalisable to more complex PINN/ for more complex problems.
- Find other areas where PIML is used and look for improvements to implementation for both:
  - Unsupervised forward problems and
  - Reverse problems.

1.3 Objectives

In order to achieve the more general aims of the project, the following objectives were set. These are more measurable and short-term focused - in part due to the unknown nature of what research will be carried out after the work on adaptive sampling.

- Implement a uniform sampling method to a PINNs solver for a benchmark PDE.

- Recreate results from literature for validation / baseline data on performance of PINN.
- Implement an adaptive strategy for sampling of collocation points aiming to improve accuracy for a fixed computational cost.
- Attend the Computational Science and Engineering conference 'SIAM' to gain thorough understanding of work being done on PINNs
- Define a research plan for an alternative area of improvement after adaptive collocation of points.

Chapter 2 : Literature Review

[Note: Relevant literature in the context of PIML has been studied as part of the work done in the first 6 months of the PhD, especially for the purposes of catching up to the state-of-the-art PINNs and finding an area of interest. However, I would like to wait until after the SIAM conference in February/March to write an in-depth literature review (See chapter 4 for the in-depth plan) for a couple of reasons. Firstly, it is expected I find at least one other area of research beside adaptive sampling that would also be an important part of the PhD and therefore of the literature review. Additionally, I expect to also have a much better idea of the current climate surrounding PINNs after the conference, so even the broader part of the literature review will probably be much better informed after attending it. This would be particularly important for this PhD compared to others - as the field of ML is so very rapidly evolving due to its popularity, I don't want to critically evaluate in too much depth older papers if follow-up work on them might occur in between now and the transfer in July.

The fact that a large if not the majority of the writing of the lit review will therefore not be done before March is taken into account in the research plan (see chapter 4), and hopefully enough time is allocated for this writing. For these reasons the literature review presented here is not very thorough. The main intention is to list some indicative papers of Machine Learning (ML) and on Physics Informed Neural Networks (PINN) design that give some historical context; and to note the more recent publications on PINNs adaptive sampling that have already been useful in deciding the project direction, as to give important context to the current work.]

Section 2.1 of this review will cover history, the types of problems data driven methods have been used for and necessary foundational knowledge on how these methods work. 2.2 will explore the role of PINNs within the current literature, highlighting original work being done and listing some indicative papers of note. In the last section, 2.3, the papers most related to the current work on adaptive sampling are discussed.

2.1 Machine Learning

Machine learning (ML) methods are used to try and solve a variety of problems. Historically, the main uses of machine learning methods have been in pattern recognition and classification, with [1] book already exploring classification methods based on data analysis being published in 1965; where the potential of ML was limited by the computational technology of the time. More recently in the late 90s, used for correctly identifying characters from images - here the problem was recognising handwritten print [2].

Currently, traditional data-driven methods are very good at those problems. More recently than this, interest has been in assessing potential of these methods to also solve scientific and engineering problems. In fluid dynamics, ML methods have been used for solving PDEs (see [3] for a comparison between ML and FEM methods) and for different types of modelling (surrogate CFD models, reduced order models) from biomedical applications ([4]) to meteorology ([5]. This sort of research is highly important to both the application areas as well as the progress of ML methodology in general, as code has to be written from scratch to address specific problems. Much can be learned from these individual papers, but the amount of literature is so vast that review papers such as [6] become incredibly useful for helping understand how the current literature addresses the problem of applying ML to fluid dynamics.

2.2 Physics Informed Neural Networks

For this PhD the interest lied principally with unsupervised methods, as these circumvent the need for data. In many of the science and engineering applications of interest, generating this data would require use of traditional methods based on first principles. This can be expensive and even when database of these results already exists requires models to be generalisable. There are still occasions were ML can be very useful in these cases; such as in medical environments where the issue would be that decision might have to be made much faster than a high-fidelity model from first principles could produce a solution. However, unlike in classification problems, in fluid dynamics and other sciences there is a wealth of information available, which physics-informed machine learning attempts to incorporate into the traditional data-driven methods to reduce the need of data whilst still harboring the many advantages of ML. Most interesting out of these is the case of unsupervised methods that, not requiring any data at all, could be used in isolation and still produce realistic solutions. In order to do this, a priori knowledge of the problem is utilised, such as information on boundary conditions or what physical equations govern the problem's physics. This approach vastly reduces the quantity and quality of data needed, and has been a very popular subject of study following [7]'s publication on "Physics Informed Neural Networks". In the years following that paper, variation of the original PINN have been used to solve a multitude of different problems. [8] offers a great in-depth analysis of many of these variants. This extensive review discusses the contributions of over 200 articles, covering the entire range of relevant literature on PINNs: from books on the method's fundamentals ([9]), to alternative ways of enforcing the physics (Such as the hard constraints implementation in [10]) and variations focused on very specific applications ([11] looks at hemodynamics applications and validates results against CFD benchmarks using a completely unsupervised approach); to good summary of different software packages. Additionally, the current challenges of PINNs from theoretical to implementation difficulties are summarised, as well as the areas of future work that can be done to address these.

2.3 Adaptive Sampling

One area of improvement identified was the positioning of collocation points. In other numerical methods requiring meshes, the positioning and shape of a mesh is often crucial. Although ML methods are meshless, PINNs utilise collocation points distributed throughout the domain to enforce PDEs at that point - for example to enforce conservation of mass and momentum to model blood flow in [11]. Some work such as [12] attempt to sample the domain in order to increase computational efficiency. Even more recent work by [13] begins to tackle this issue and compares a variety of uniform sampling methods with 2 adaptive sampling methods. Their results on various benchmark problems using uniform sampling methods offer a great dataset I can attempt to validate against, thus making this paper useful as a starting point for attempting other novel adaptive sampling strategies.

Chapter 3 : Progress To Date

In this chapter the main progress done to date will be covered in mostly chronological order. The main purpose at the early stages of the PhD was to learn the state-of-the-art of machine learning methods through a hands on approach. For this reason, the focus so far is not on making original contributions but on being able to implement machine learning methods to solve simple problems. In section 3.1 the main objective was to build and implement a neural network that could attempt to solve a trivial problem.

Having an appreciation for the basics, the next objective was to look at a more relevant problem. Simultaneously, a more structured approach to learning was taken by also signing up to an online course (section 3.2. A look was taken at publicly available PINNs code present in the literature based on [14] was used [1]. This code was then used for an investigation of how different parameters of a PINN affect the solution, as a learning exercise discussed in section 3.3.

An interesting direction for the topic of research was to look at the implementation of adaptation methods for the process of distributing collocation points - points where the physics is enforced throughout the problem domain. To move in this direction, very preliminary investigations were done on how different ways of distributing these collocation points would affect the solution process. This work is then covered in section 3.4.

3.1 Toy Problem

At the start of the project, the wide breadth of the topic meant it was hard to tell exactly what kind of problems I would be focusing on further into the PhD. It was therefore thought optimal to begin by looking at developing some general practice on coding neural networks. To achieve this practice, I first looked at PyTorch, a framework for coding machine learning methods in python. The PyTorch official website contains some tutorials with example problems that allowed me to learn both how to use PyTorch to code a simple neural network, and how these neural networks are trained to solve problems.

The first sample problem involved simply fitting a polynomial to the line $y = sin(x)$. The network was first implemented by using the popular science python library NumPy

[1](available from https://github.com/okada39/pinn_burgers)

to perform the same actions a more complex network would carry out. First, the output of the polynomial was calculated with the current weights, and the loss (difference between the output and true value). From this, the gradients of the weights with respect to the loss was obtained and used to update the weights in the direction of quickest minimisation of the loss. Looping these steps thus minimised the loss function until the polynomial was trained to predict $sin(x)$. This network was enhanced then by adding PyTorch functionality to the different aspects in turn. First, instead of manually computing the gradient of the loss, PyTorch modules were imported that utilise automatic differentiation to allow for the gradient to be computed even as the network is made more complex with more nodes and/or layers. Tensors (PyTorch's version of NumPy arrays) replaced the arrays, allowing for GPU computations. Lastly, PyTorch was used to define the optimisation method and loss functions to be used. The final result was a fully connected network capable of approximating $y = sin(x)$.

Whilst this was a good starting point to get familiarised with the basic usage of PyTorch, I didn't really have the tools to understand or apply the more complex Physics-Informed-Neural Networks; like for example the work of [7]. However, by the end of this work I had set up a coding environment and version control for my coding projects via Visual Studio Code and GitHub that would help with data management; and more importantly, had a more clear idea of how to best continue learning the necessary coding skills for the project.

3.2 Python and Tensorflow 2 Training

At the same time, I had been attending some more informal weekly python programming tutorials run for the CDT by Patricia Ternes. In these we tackled some 'programming challenges' that helped us learn or revise useful techniques for general scientific computing. These ranged from tips to improve readability and efficient computation (with techniques like list comprehension) to practice using scientifc python libraries like NumPy for data processing and implementing randomness. But as mentioned at the end of the previous section, I had also found a more appropriate way of learning the essentials of machine learning for my project. I observed a lot of code on PINNs specifically used TensorFlow2, and I was recommended the coursera specialisation on TensorFlow2 for Deep Learning. Prerequisite knowledge for this included "knowledge of general machine learning concepts (such as overfitting/underfitting, supervised learn-

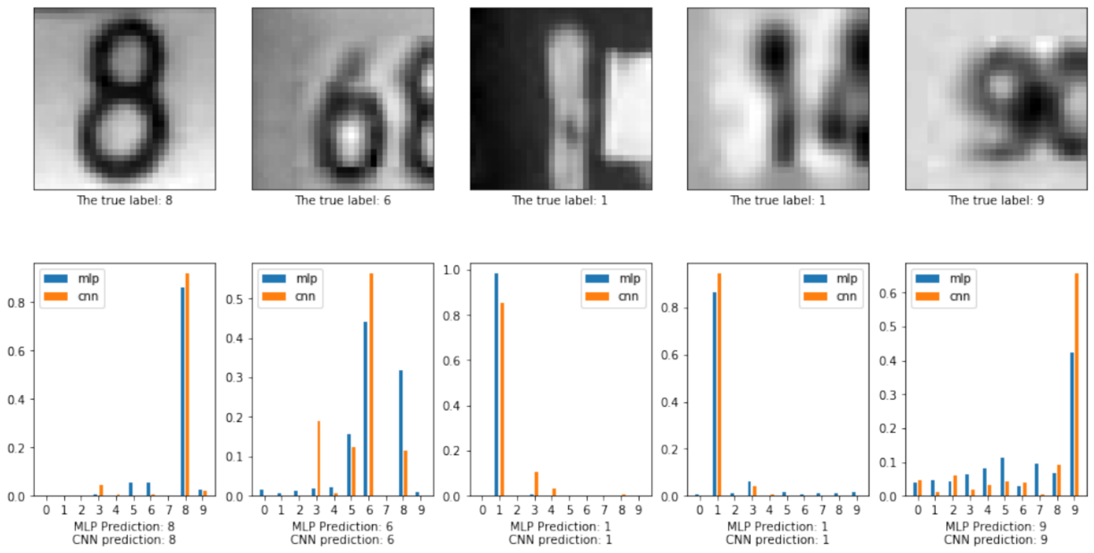


Figure 3.1: Above: Image input to classifiers; Below, bar graphs showing the perceived likelihood of the image matching each number, with the networks' guess below.

ing tasks, validation, regularisation and model selection), and a working knowledge of the field of deep learning, including typical model architectures (MLP/feedforward and convolutional neural networks), activation functions, output layers, and optimisation". A foundational course offered by Google "Machine Learning Crash Course with TensorFlow APIs" was completed in order to catch up to the prerequisite knowledge required.

Then, the first of 3 courses forming the Tensorflow2 for Deep Learning specialisation was undertaken, "an introduction to Tensorflow2" with the following main learning objective:

- "To learn a complete end-to-end workflow for developing deep learning models with Tensorflow, from building, training, evaluating and predicting with models using the Sequential API, validating your models and including regularisation, implementing callbacks, and saving and loading models."

This course involved learning through a mix of pre-recorded instructional videos, tests and coding tutorials; with a ”Capstone Project” assignment due at the end. This project involved developing both a feedforward MLP network and a CNN model for classifying images from the Street View House Numbers (SVHN) dataset. These two were coded and used to accurately classify unseen images from the dataset - see figure 3.1.

The second course was also begun - "Customising Tensorflow2 modules", which looked at expanding knowledge by teaching how to use lower level APIs, which will be necessary for creating and adjusting custom PINNs networks. The official learning objective is described as:

- To "deepen your knowledge and skills with TensorFlow, in order to develop fully customised deep learning models and workflows for any application. To use lower level APIs in TensorFlow to develop complex model architectures, fully customised layers, and a flexible data workflow. To expand your knowledge of the TensorFlow APIs to include sequence models."

3.3 PINNs for Burger's Equation - Hyperparameter Investigation Report

To get hands on experience with PINNs, github code solving Burger's equation based on [7]'s work was pulled and modified. The idea was to play with hyperparameters of the network to get hands on experience of effects of different parameters of network and a feel of how big these effects were relative to each other. To undertake a proper quantification of effectiveness of network, first some ground truth data had to be created to compare the results of the network to. As burger's problem is relatively simple, this was done via a Finite Difference Method (FDM) approximation.

3.3.1 Generating FDM Ground Truth

The objective was to have an accurate, fine grid to compare the NN solution to. To compute the answer to Burger's equation, it was re-written in ODE form using a central difference approximation. An ODE integrator python library from SciPy was then used by looping through space and integrating the ODE in time to generate a square matrix of $u(x, t)$ at every point in space and time in the domain ($-1 < x < 1$ and $0 < t < 1$). To quantify the discretisation error of the FDM grid (whether it was fine enough), carried out a convergence study by looking at pointwise discrepancy (Quantified by the L_2 error, see eq 3.1) between grids increasing in resolution (see figures 3.3a, 3.3b, note 6400 means L_2 error between 3200 and 6400 grid). These were

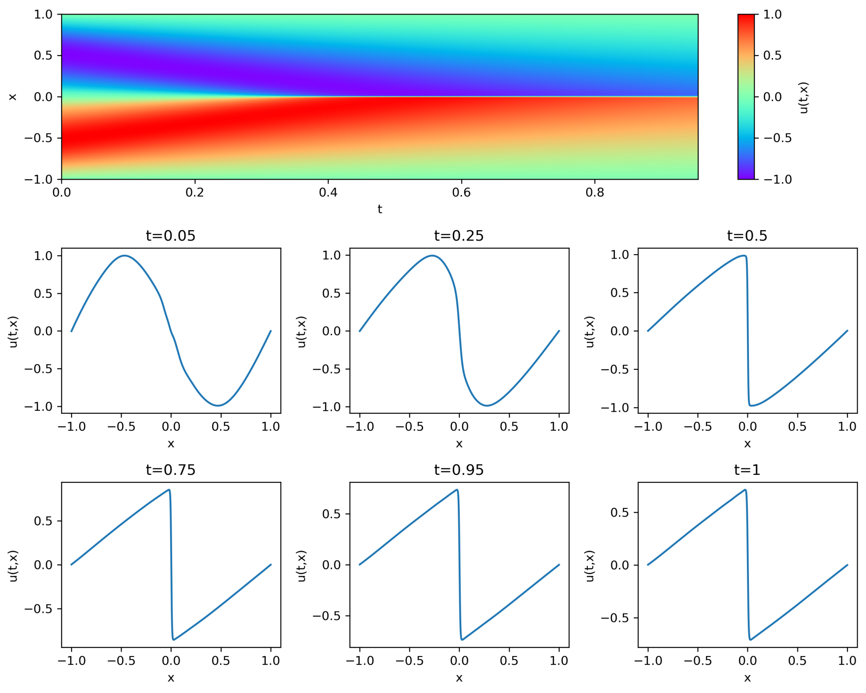


Figure 3.2: Burger's Equation solution using 8000 training points and a 3 layer network shape $32 \times 16 \times 32$. Above: Contour plot of u(t,x); below, 2D slices of the domain at different t.

compared by restricting the higher resolution grid. Interpolating the lower resolution grids into higher resolution was also considered but would've been more time-consuming and the grid was determined to be adequately fine enough at 6400 points from the mesh convergence study. As the solution data was obtained through the study there was no reason not use the highest resolution grid obtained.

$$L_2 = \frac{1}{n}\sum_{i}^{n}(u_{i,c} - u_i)^2 \tag{3.1}$$

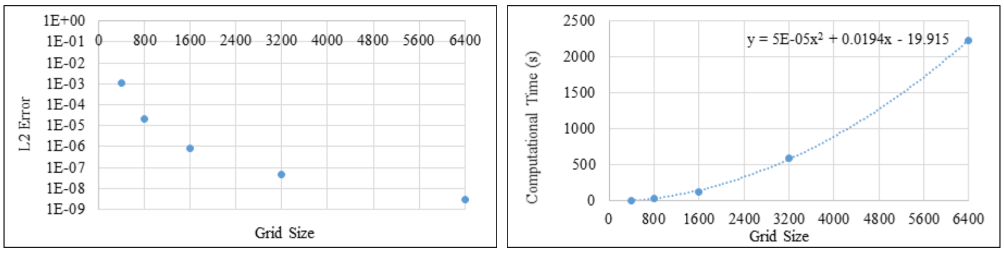
Where:

n = Total number of points,

$u_{i,c}$ = Velocity at point i in restricted grid,

u_i = Velocity at point i in lower resolution grid,

For comparing the performance of the PINN, it was necessary to look at some quantitative form of error as visual inspection was not good unless results are very wrong - most network parameters capture the burger's equation shock general shape but it's hard to spot small deviations or errors in magnitude without taking a detailed look at slices throughout the domain. It is also very hard to compare the more complex / accurate networks as the differences between their solutions and the ground truth solution become too subtle (See figure 3.2 for an example visual solution). Therefore, the main way of comparing the accuracy of networks was to again look at the pointwise error. For the network, no restriction was necessary, as once trained the network could be used to predict the solution onto a grid of any size. This allowed to simply subtract the matrix of ground truth solutions from the NN prediction and take the mean error over the domain.



(a) FDM Solution: L_2 Error vs Grid Size, noting grid size indicates length of square array ($N \times N$). Note log scale on y-axis.

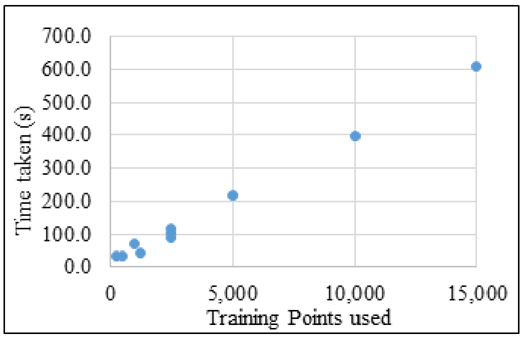
(b) FDM Solution: Computational time taken vs grid size. Dotted line is quadratic best fit approximation (function shown).

Figure 3.3: Grid independence study of FDM solution.

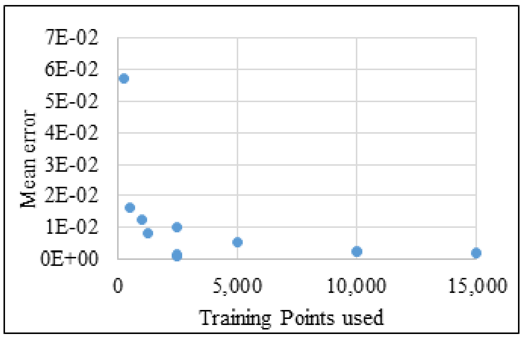
One of the investigations looked at was how changing the viscosity of the problem would affect the network's ability. For this case, a separate solution had to be computed with the different viscosity. Instead of repeating the grid size sensitivity study, the same grid size of 6400x6400 was deemed sufficient and used. For lower viscosity ($\nu = \frac{1}{400\pi}$), run time increased a substantial amount (from 37 minutes to 54 minutes for the finest case, a 46% increase). Increasing to a much higher viscosity ($\nu = \frac{1}{\pi}$) led to a case too viscous for a shock to form, so the case was discarded.

3.3.2 Hyperparameter Investigation

3 main hyperparameters were investigated: Training point number, shape of neural network (mainly node number) and convergence criteria (factr). Then viscosity was also changed, changing the equation to be solved. The training point investigation was then repeated for the new viscosity, comparing against the other ground truth



(a) Training Points vs error



(b) Training points vs time

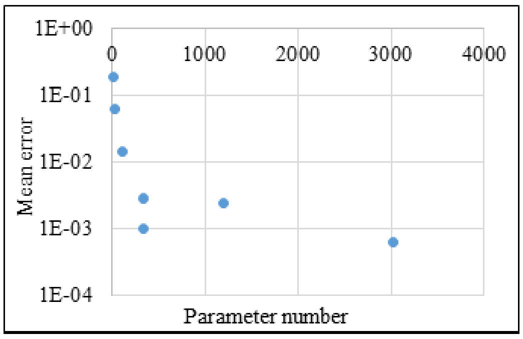
Figure 3.4: Training point number investigation, for viscosity of ($\nu = \frac{0.01}{400\pi}$) and the default NN shape ($32 \times 16 \times 32$).

solution. The max number of steps was limited as a compromise due to time; which is worth noting as it could have limited the maximum possible accuracy for training with slower convergence. Regarding the PINNs default optimisation methods, the default L-BFGS-B algorithm was kept as the optimiser for the network; and the default 'tanh' activation function and learning rate were also left unmodified. With these settings, it was possible to look at parameter trade studies and their effects on mean error and time taken.

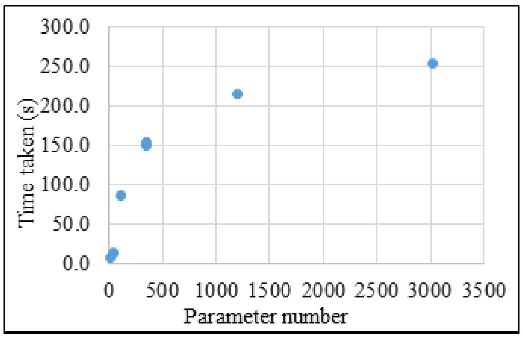
Training points: At this stage didn't have a way of automating runs to efficiently do re-runs of experiments. There was an anomaly run 3 times at $N = 2500$ where, despite having less training points than other runs, was much more accurate than more dense, slower simulations. These results can be seen in figure 3.4. Generally, increasing number of training points linearly increased time taken for network to train. Increasing the points decreased the error as expected, with the increase in accuracy requiring more training points.

Network shape: Default shape was 3 dense layers of 32, 16 and 32 nodes respectively. This investigation was carried out by halving the number of nodes and looking at the effect on the quality of the solution. [7] paper mentions they utilised 6 layers of 20 nodes, so this was also looked at. In order to quantify this, the number of parameters (total connections between nodes) was plotted against the mean error. For reference, the 6x20 case had 3021 parameters compared to the default case's 1201 parameters. Results can be seen in figure 3.5.

For figure 3.5b, time taken seems to approach a finite number with increasing para-



(a) Number of parameters vs error



(b) Number of parameters vs time

Figure 3.5: Network shape investigation results. Shapes followed 2N:N:2N shape with the exception of the 3021 point which had 6 layers of 20 nodes each. Note log scale on error graph.

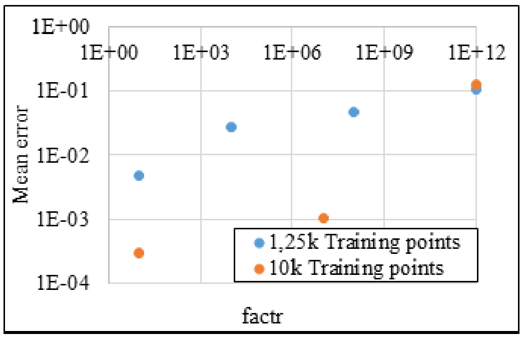
meter number, but might be that the higher layer '20s' case is more efficient computationally, giving it the wrong shape. For a relatively small increase to time taken, the 6 layer case has lower error than the other cases (note the log scale means this is lower than may appear).

NN Convergence Criteria: Defined as "factr" within the code, it was a unique variable to this implementation of PINN that I was trying to understand. It seems to dictate at what residual the solutions would be considered to be converged. The results of adjusting this parameter can be seen in figure 3.6.

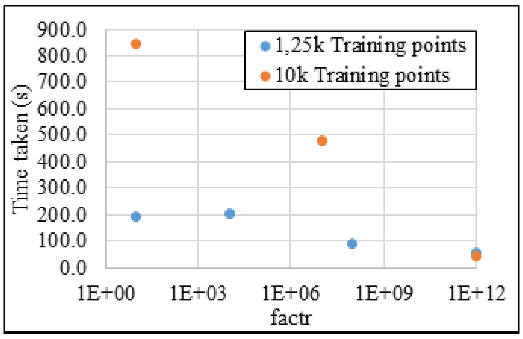
The effect of factr was looked at for both a high and a low number of training points, to better understand when it plays a factor and whether its dependent on others. Looks like a linear relationship in time, but its harder to tell for the mean error. However, it's clear that with more points gradient of time taken higher, as a result sacrificing factr yields much greater time saves. At the same time, more training points means error still lower for modest factr. Moving on, the 'modest' factr value of 1E7 was used, as it produced suitable accuracy for the hyperparameter studies without compromising the computational cost too much.

Viscosity: To look at viscosity, again the default network shape was used. The mean error and time were compared between cases as the number of training points used was varied.

It was observed that the PINN network took a similar amount of time to compute answers for both cases, but had a lower accuracy for the lower viscosity case ($\nu = \frac{0.01}{400\pi}$).

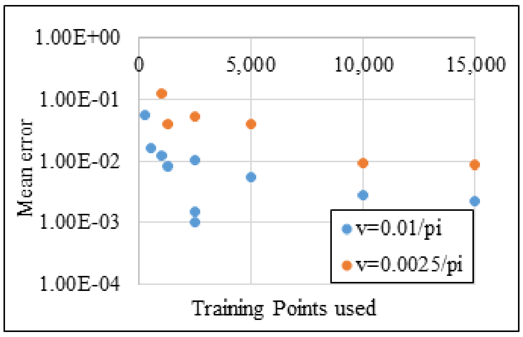


(a) factr vs error

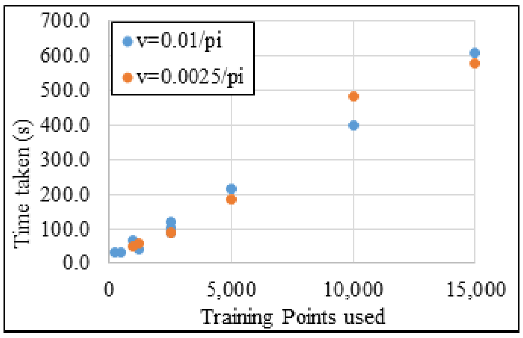


(b) factr vs time

Figure 3.6: Convergence criteria (factr) investigation (lower number = higher accuracy). 1E1 = highest accuracy, 1E7 = modest, 1E12 = low accuracy



(a) Error vs Number of Points



(b) Time vs Number of Points

Figure 3.7: Results for different viscosity cases

Overall the network seemed quite robust and didn't fail to produce a solution unless it had very very few points - even the simplest NN shapes with 3 layers of 8, 4, and 8 neurons were able to predict the burger's equation shockwave. By calculating the mean error across the domain, it was possible to compare how the accuracy was affected by the various hyperparameters. Plotting the results of this, could easily see the trade-offs between computational cost and accuracy for the different parameters and confirm the expected behaviour: generally, more complex and bigger networks provide increase to accuracy at an increased computational cost - with increases to accuracy requiring exponentially more resources below a certain threshold.

Overall, these sets of investigations were really beneficial for getting a grasp on the basic concepts learned at the same time through first coursera course, and how they apply to PINNs specifically. Having to modify the code directly also meant I learned exactly how these hyperparameters were implemented in python and how the machine learning method was coded in practice. This would be incredibly useful later on for knowing how to modify the code in order to investigate biasing/adaption.

3.4 PINNs for Burger's Equation - Biasing Investigation Report

Having a better idea of how PINN network worked, I wanted to look at changing the fundamental method employed by this PINN in ways that might bring about benefit. Namely, to investigate whether the way collocation points are distributed influenced the cost and accuracy of the solution. If it did, this could warrant an investigation into smart methods of distributing the collocation points - adaptive collocation methods. The first step for this was to look at how biasing the points location would affect the results. A very simple bias towards the center of the domain was tested, and results seemed to indicate a potential benefit could be obtained from this. A more complex shape was given to the bias to see if the effect could be recreated. Lastly, it was noted that both the sampling of points and the ML methods have a random nature, and I wanted to quantify the effect of the randomness of the sampling of points on the variance between runs. To do this, I looked at the variance in error through a series of bias studies.

3.4.1 Simple Biasing

For the first look at biasing, the $32\times32\times32$ NN architecture was chosen. As the number of points used was predicted to also influence how much bias would have an impact, various numbers of points were tested too. Did a preliminary very simple approach to bias towards middle (which can be seen in figure 3.8) to see if had an effect. The biasing was done by dividing the domain in 3 parts, and adjusting the number of points distributed within each area according to the desired bias. This random sampling was relatively trivial; a random number generator was used to generate a $[2 \times N]$ vector of floats between 0 and 1, where N is the number of points. By multiplying this vector by the domain range, an (x, t) vector of random coordinates is produced within that range. When biasing into 3 areas, the number of points in each area N is adjusted as is the range.

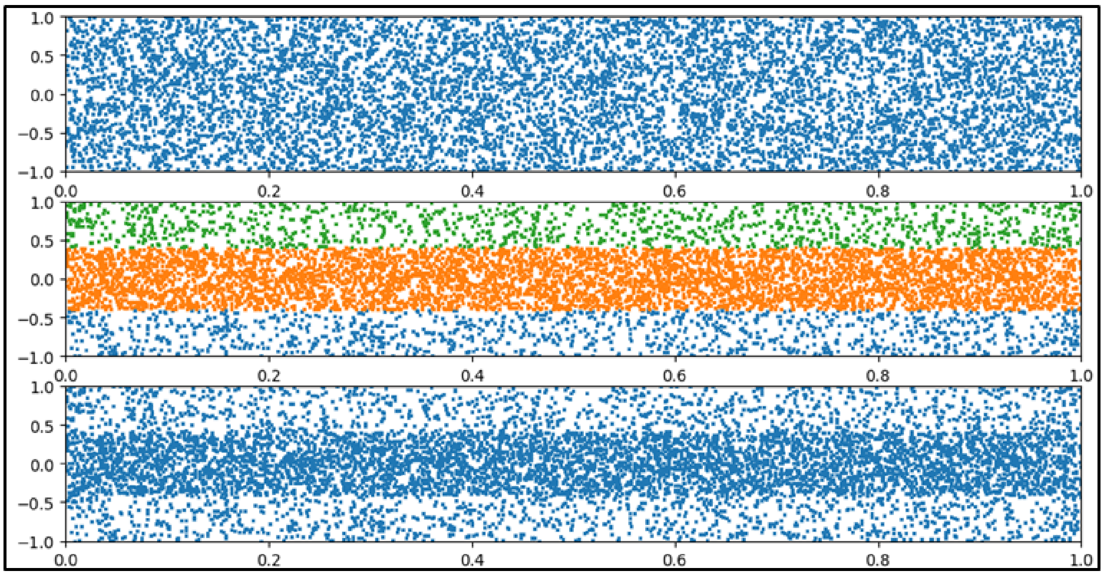


Figure 3.8: Random distribution of points throughout the domain at the top versus the distribution using a simple bias towards the center.

To quantify this bias, it was defined as :

$$\text{Bias ratio} = \frac{N_{Bias}/N_{Total}}{A_{Bias}/A_{Total}}, \tag{3.2}$$

Where:

N_{Bias} = Number of points in biased section,

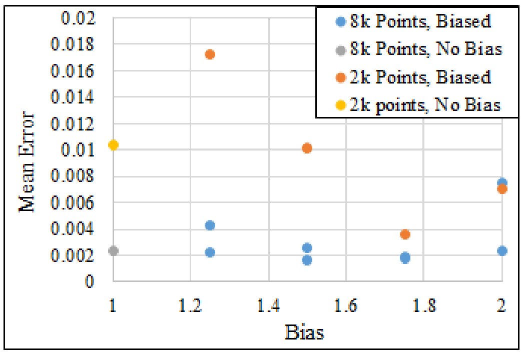
N_{Total} = Total number of points,

A_{bias} = Area of biased section,

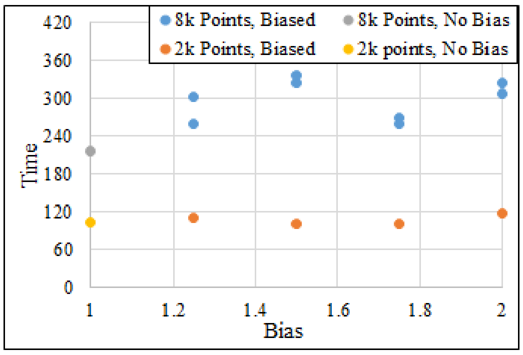
A_{Total} = Total Area

So if 40% of the points were in 40% of the area, bias = 1 = no bias. If 80% of the points were in the same area, the bias = 2.

To test the effect of this, repeated runs were made for every bias. For each run, independently of the number of points used, the network was used to predict the velocity field on a grid the size of the FDM solution grid. This way the difference between the predicted solution and the ground truth FDM solution could be calculated at every point. The average error in u was then taken. This average error and the time taken for the computations was then stored.



(a) Mean error vs Bias.



(b) Time vs Bias.

Figure 3.9: Results of the simple biasing.

Observing the results in figure 3.9, it can be quickly observed that increasing the number of points significantly increases the accuracy of the solutions, as observed in the hyperparameter investigations, despite the different distribution of collocation points. A small improvement can be seen with regards to accuracy for the 8k points case, but more significantly, for the 2k case a more significant improvement can be seen when the bias is between 1.5 and 2. These preliminary results were interesting enough that it was thought worth repeating with a less simple biasing method, for which the number of runs could be repeated more often to ensure there results were statistically significant.

3.4.2 Refined Biasing

Next step was refining the bias area. This was less trivial as it was not longer possible to populate the non-rectangular bias areas with the methods described earlier. The bias areas were split into triangles and then a slightly more complex function was used to generate points in the triangle given the vertices of this. Functionally though, the biasing worked in the same manner; with the bias being described by assigning the appropriate number of points in each bias area and distributing them randomly. The more complex shape can be seen in figure 3.10. Additionally, at this point the PINNs code was modified so it would save the prediction results and allow for looping of different parameters such as number of training points. This allowed repeated runs to be run automatically for different investigations, increasing the reliability of the results. [Note: For investigations between February and June transfer report looking at Regular Sampling and comparing to literature (See Chapter 4), I'd like to look at plots of Solution vs Number of Repeated Runs, to look at how these results converge and properly assessing whether the number of repeated runs is enough. However, at this time, 5 repeated runs were done as a compromise due to the time these runs would take.]

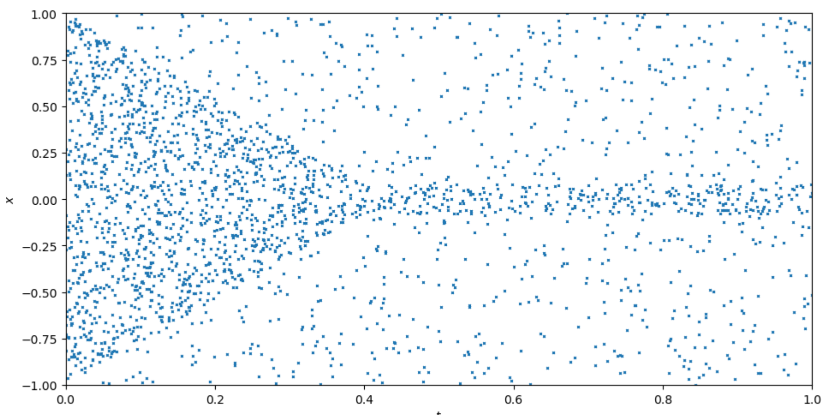
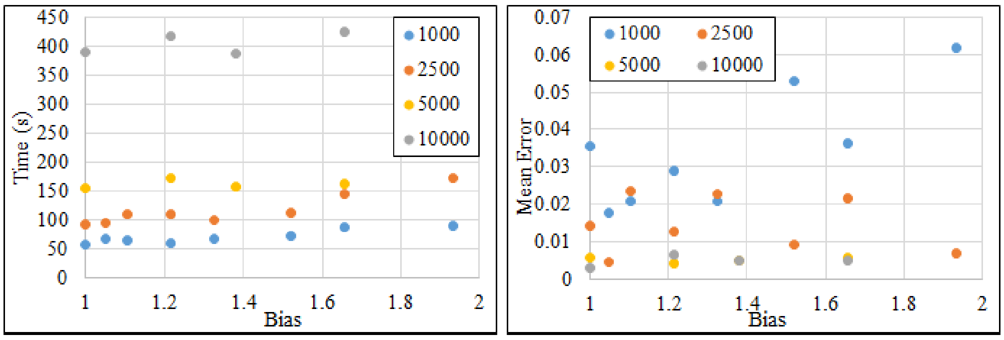


Figure 3.10: Collocation points distribution into more complex bias shape

Key results of the bias investigations are shown in figure 3.11.



(a) Time vs Bias

(b) Error vs Bias

Figure 3.11: Line graph showing results of biasing as shown in figure 3.10, with a central width in x of 0.3 and trapezium height of 0.5 in t.

For this bias shape it is harder to observe clear improvements to the accuracy from the introduction of biasing. The usual expected patterns observed in section 3.3 can still be seen: increasing number of points increases accuracy up to a point at the cost of increasing computational time. Introducing bias didn't seem to increase computational time very much, trivially compared to the effect of changing number of training points.

The effect of biasing at high number of training points was negligible as expected - as with enough points even non-refined areas will have more points that non-biased cases with less points. For the cases with lower number of training points (1000 and 2500), some small improvements were observed but not in a consistent way - the 1000 case saw small improvements with very small biasing between 1 and 1.4, whereas for the 2500 case improvements to accuracy seemed to occur randomly with bias (most likely requiring more repeats to better investigate a pattern). It is worth nothing that choosing the area of refinement was done by manually observing where the highest velocity gradients existed in the solution. It is hard to know whether this would be equivalent to the ideal area to refine in order to increase accuracy or reduce residuals, and this could've affected the helpfulness of the biasing.

3.4.3 Variance

Before continuing on to further work, one observed issue was the variance between runs being so big it made it hard to spot the patterns (See figure 3.12). Hence last work done in December was looking at the impact of the random distribution of points compared to the random nature of the ML method. Quickly checked how much the error varied when keeping the points in the same place (See figure 3.13). From this particular example it seemed like with constant settings, the random distribution of the points seems to increase the variation quite significantly, with individual runs seeming to be distributed much closer to the mean.

To do a more thorough investigation of this, a slightly more sophisticated way of checking this was planned by looking at the variance. The variance was computed by evaluating the standard deviation of the errors, and how this changed as the number of collocation points was increased was observed for different biases. This was done for 2 cases: in the first case, the collocation points were sampled randomly according to the bias specified every re-run; in the second case, the collocation points were sampled randomly once and then the same distribution was re-used by setting a seed for the distribution of points. The results of this can be seen in figure 3.14.

Expected fixed points to reduce variation (solid line below dotted line) and variance to reduce as number of points used increase. The first expectation was generally observed to be more true as the number of points was increased, but was not true across all cases. Possibly, the variance was not calculated from enough points. Additionally,

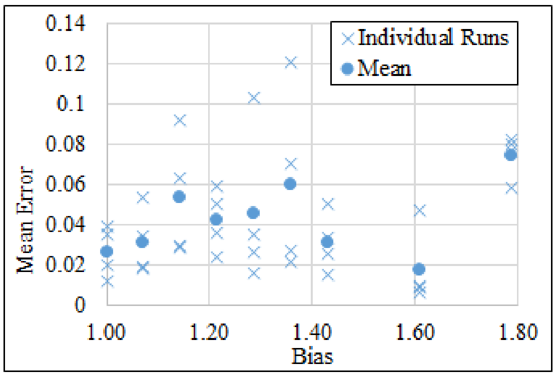


Figure 3.12: Error vs bias runs showing the spread of individual run errors. For the 500 Points case seen in figure 3.11

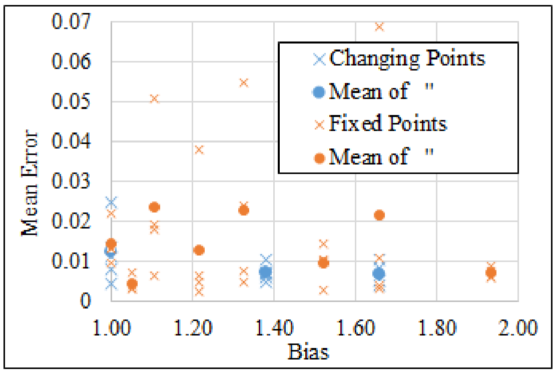
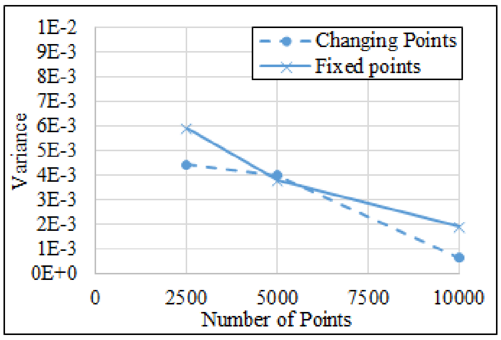


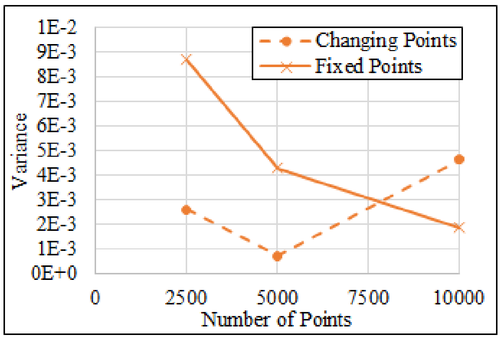
Figure 3.13: Graph showing the comparison between fixing the points and re-sampling every run, for the 2500 points seen in figure 3.11

at the outset it was assumed that the variation due to the method would be consistent, and that therefore removing the randomness of the points sampling between runs would allow it to be observed. However, it is possible that if the distribution of points that was fixed happened to be relatively bad this could impact not just the accuracy of the network but also the variation produced by the network between runs. This should be less of an issue as the number of points is increased as the effect of distribution and bias decreases, as observed previously.

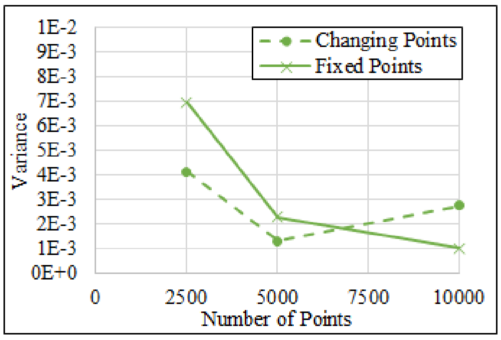
Overall, high variance due to the use of random sampling every run was observed. This calls attention to the need of using uniform methods such as Latin Hypercube Sampling (LHS) instead of simply random sampling, even before looking at adaptive sampling. This would at least to help make it easier to spot patterns by reducing the



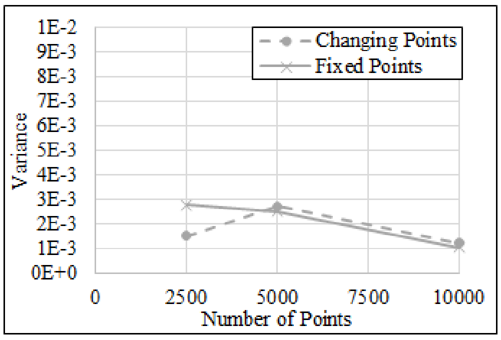
(a) No bias



(b) Bias of 1.22



(c) Bias of 1.38



(d) Bias of 1.66

Figure 3.14: Effect of re-sampling points every run through variance vs training points for increasing biases.

variance between runs. Thanks to the fact a benchmark case was being analysed a ground truth solution is available so it is possible to see that some of the solutions output by PINN are good even with random sampling. However, this will not be the case for the vast majority of applications. Because of this its hard to justify the use of completely random sampling as multiple re-runs would be needed due to the variation in accuracy from case to case. Additionally, LHS and other similar sampling methods will potentially be easy to implement and not add too much to the computational cost, as they are established and well understood methods, so there is a high likelihood that optimised code libraries already exist I can import.

3.5 Next Steps

Intuitively it makes sense that some collocation points will be better than others at training the network, but there are many ways this could be looked at. A more complex approach suggested by supervisors was evaluating what points are most useful and refining near those areas in adaptive method by distributing more points every certain number of steps throughout the training, instead of distributing them differently around the domain from the start. In a real application, this would be the necessary approach anyway as the areas of interest would most likely not be known from the outset. Thus, having learned how to adjust the distribution of collocation points within a PINNs network, the next step was to develop a 'smart' sampling method. At this point the recent paper [13] was looked at. It tests both non-adaptive methods and it's own suggestion of adaptive methods. Moving on, I would continue to work investigating and recreating results of work such as that, whilst planning exactly how to implement an adaptive sampling method (more detailed plan in chapter 4). Overall, thanks to the investigations carried out and the courses undertaken, I can understand most aspects of how the PINN code I've been working with works and I am more confident that on completion I could even modify more complex aspects of PINNs or even develop my own more complex functionality in order to incorporate adaptive sampling.

Chapter 4 : Research Plan

4.1 February to June

February:

Continuing with the work done so far, the first step is find and use code to triangulate domain - a way that is scalable to multiple dimensions in future may be useful. Discretising the domain in this way will allow for measurements and fine-tuned distribution of collocation points later on.

As the FFPR is written in the format of the transfer report, once feedback for it is received, make relevant adjustments to sections that need it so the document can serve as a first draft of the transfer report. If any additional investigations/validation tests for work done so far would be necessary, aim to finish these within the end of the month before continuing onto other work.

Plan investigation into regular random sampling - this will be the baseline adaptive sampling method will be compared to. Recreating the relevant results of research such as [13] would also be necessary, as the work will have to be compared to work from literature. Have to read into the strengths and weaknesses of different regular sampling methods (Latin Hypercube, ...) and choose between them.

I will also be starting the "Deep Learning" master's level module - ongoing until June. I also plan on attending relevant seminars throughout the semester organised by the following groups: the Leeds Fluids & MHD seminar, SciML Leeds talks, the AI^2 Forum, and the CDT internal seminar; as well as any other relevant talks on PINN I find.

March:

First week of March I will attend a Computational Science and Engineering conference with many relevant talks on SciML. A main objective at the conference is to find other areas of interest within PINN to work on next. To stay up to date with literature, I also want to make sure I subscribe to emails from relevant journals on Web of Science so I can check out daily what's been published and save papers relevant to me as they come out.

Make clear plan on how adaptive sampling strategy will work. Things to consider:

how to know what areas to refine, how refining will be done, how much to refine by. Should try to write this code so it can easily be modified to work for other network. Should study similar strategies from the literature - looking at how this has been done outside of ML too, such as in adaptive FEM.

Code and run investigations on regular sampling methods throughout month. If these can be done in background/overnight, could begin coding parts of adaptive sampling.

April:

Finish investigations on regular sampling methods, make comparisons to literature where relevant. Probably analyse and write up discussions/ prepare figures for report. At this point should begin coding adaptive sampling method if haven't already.

As a follow up to the conference, for work in other areas of PINN I might want to explore, should start thinking at this time about what training/skills might be necessary for that research. Perhaps taking some time to look through new literature in the areas where I could contribute.

May:

At some point in May would want to have a working way of sampling collocation points adaptively based on criteria such as local residuals [13] or gradient-based [15]. With a working model could run investigations to compare with the investigations run in March.

At this point, thanks to the conference and literature reading on my own time, I should have more to add to the literature review written for the FFPR. It would be a good idea to aim to have updated this by the end of May instead of leaving all of the writing of the transfer report for June.

Will also be revising for COMP5625M Deep Learning exam.

June:

Realistically will spend the vast majority of June focusing on the transfer report - writing, preparing figures, and running any outstanding simulations. However, in spare time could perhaps continue reading more on current state of PINN, or continuing training on TensorFlow.

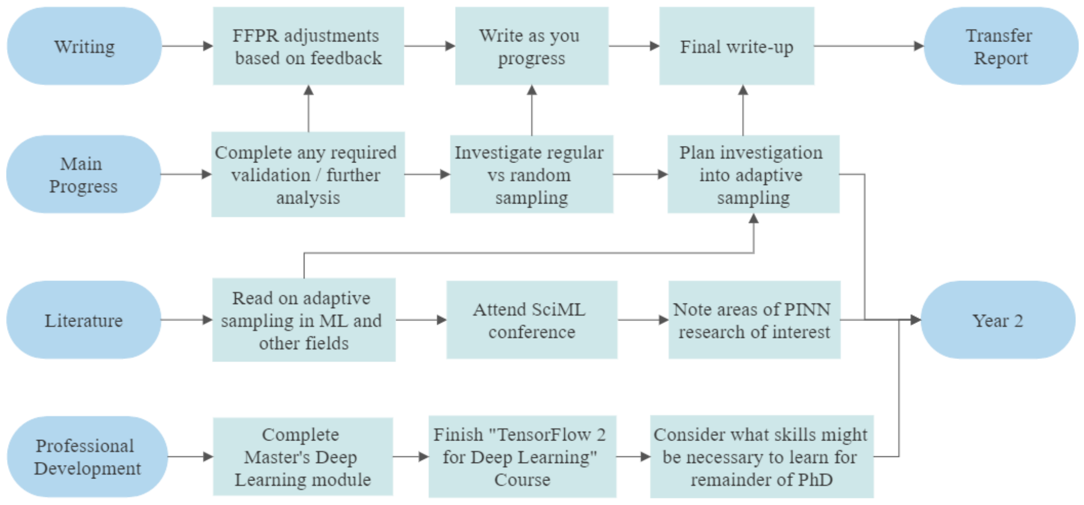

Writing
FFPR adjustments based on feedback
Write as you progress
Final write-up
Transfer Report
Main Progress
Complete any required validation / further analysis
Investigate regular vs random sampling
Plan investigation into adaptive sampling
Literature
Read on adaptive sampling in ML and other fields
Attend SciML conference
Note areas of PINN research of interest
Year 2
Professional Development
Complete Master's Deep Learning module
Finish "TensorFlow 2 for Deep Learning" Course
Consider what skills might be necessary to learn for remainder of PhD

Figure 4.1: Flowchart summarising the plan from February until the transfer report deadline.

4.2 Second Year - July to December

After transfer report, extend current work on adaption by implementing new approaches. Extend to other benchmark fluid dynamics PDEs if a possible improvement has been observed on the Burger's case. As the work on adaption is carried out, in case the results aren't as promising as hoped, continue to look out for other possible areas of interest still related to the research question - i.e. looking to improve specific aspects of PINN methodology / implementation in general. If any skills need to be learned (programming language, etc) this should be a main focus over the summer months following the transfer report submission. Once sufficient work has been done on adaption, if any original contributions of note have been made, my primary objective would be to carry out any additional validation/ studies necessary so the work can be submitted for review/publication.

Outputs

In summary, these would be the planned outputs from this 6 month period:

- Develop and test efficacy of adaptive sampling of collocation points in PINN
- Consider training skills needed for researching other areas of PINN

- If sufficient original contributions have been made in adaptive sampling, carry out necessary work for it to be worthy of submission for publication

4.3 Second Year - January to June

Work on improvement of PINN networks. If significant findings were done in the previous year, submitting the work for publication should be a priority.

Outputs

In summary, these would be the planned outputs from this 6 month period:

- Complete draft of publication on adaption work and submit if relevant.
- Assess opportunities for developing the adaptive PINNs further (in light of both work completed and work of other researchers working in parallel).
- Reassess my progress in understanding current research developments in other areas of PINN and how I could potentially make original contributions to field between now and thesis write up.
- Develop plan for final year of PhD.

4.4 Third Year

Reassess what work can feasibly be completed and written up by end date. Adequately plan out a research plan for the last year and execute it to finish thesis before end of third year.

Outputs

In summary, these would be the planned outputs for the final year:

- Clear structure for the final thesis draft
- Plan of how to complete remaining work.
- Written up thesis.

Chapter 5 : Scope for Original Contributions

The relatively broad research statement of this PhD means that currently there is a broad range of original contributions that could be made. So far, work has been done on one specific aspect of PINN implementation, looking at the distribution of collocation points. In this chapter, I will first discuss the potential of this work on adaptive sampling; and then, by analysing what sort of contributions are currently being published, discuss the nature of other potential contributions that could be made.

A novel way of adaptively sampling collocation points could potentially bring about improvements in various forms: utilising less (or better positioned) points could decrease overall computational cost of training; or, even at a slight increase to computational cost, increase the overall accuracy of the solution. If the latter was achieved through the introduction of a novel sampling strategy, this would be a very impactful contribution to the current literature and there would be a great deal of interest in trying to generalise the results to more complex problems.

As described in the previous chapter, one major objective for the next 6 months is to better clarify what other work in PINN could be carried out that helps fulfill the research statement of this work. Doing this should also help narrow down the scope for original contributions of this PhD as a whole. The scope of original contributions for PINN research in general is quite wide due to the active interest from many different scientific communities. As a result, there are many journals that welcome original contributions to the field:

- Journal of Computational Physics (JCP): Accepts contributions dealing with ”computational aspects of physical problems” and encourages original scientific contributions in advanced mathematical and numerical modeling. The landmark paper on PINN [7] was published in JCP.

- Computer Methods in Applied Mechanics and Engineering (CMAME): Covers use of computational methods for the simulation of complex physical problems, specifically mentioning fluid mechanics as an example problem field. Would accept fluids-specific contribution to PINNs, but has also accepted contributions on the field of adaptive collocation I'm currently working on, such as [13].

- Journal of Scientific Computing (JSC): Looks for ”developments in scientific computing and its applications in science and engineering”. Has covered work such as extending PINN method to help solve PDEs [16].

- Communications in Mathematics and Statistics: Focuses on mathematical derivations of deep learning based methods, and may be more suited for contributions involving creating specific PINN formulations to solve a specific set of PDE problems, like [17].

Whilst high-level, this list still provides a good insight into what kind of contributions would be of most interest to the ML community, and helps cover the wide scope for original contributions that can still be made within the field.

Appendix A : Appendix 1

A.1 Responsible innovation plan

Consideration should be given to potentially unintended consequences of any innovation done as part of this PhD. As the research field is heavily computational, there aren't as many ethical dilemmas that might arise from human participation. Additionally, as applications of physics informed neural networks are vastly related to engineering and scientific problems, it is very unlikely that privacy concerns relating to data obtained from people would arise. However, some consideration should be taken regarding the energy use that might arise through the use of computational methods. In particular, if there is an exploration into parallelisation of PINNs methods, so that they can be accelerated by GPUs and high performance computing (HPC) systems, there could arguably be some ethical dilemmas with regards to whether the advancements made to science will offset the potential carbon footprint of running expensive computations.

A.2 Training needs and plan

General professional development on softer skills like ethics, outreach and research are carried out as part of compulsory modules.

Specific courses on relevant software have been undertaken through University's affiliation to Coursera on TensorFlow2. Additionally, the optional module ML should also be useful for this purpose.

Additional training needs should be revised in the second year of the PhD as other areas of interest within the field of PINN are explored. However, due to the rapidly-changing nature of this field of research, it is yet too early to know what other areas this PhD might focus on and therefore what training needs might arise.

A.3 Data management plan

Currently, code is written in Visual Studio Code and linked to Github. This incorporates version control to all the code I write and I periodically sync the code/analysed data/reports to github, effectively backing up the more important information. The raw results usually occupy more storage space and so are not backed up online, but if it's necessary to keep these it would be worth making local copies on either external hard disk drives or on other computers.

A.4 Expenditure plan

As it is unlikely there will be any experimental aspects to this project, it is safe to assume the majority of costs incurred will be associated with attending of conferences and seminars. The purpose of these is to become involved with the scientific machine learning community. This should be especially important in this PhD due to the quickly evolving nature of this kind of research making it important to keep in contact with other researchers. For example, this could help know what other's are working on before their work is published, helping avoid the very real risk of working on something that's already been done.

Some allowance should be made for possible costs of training.

References

[1] N. J. Nilsson, *Learning machines : foundations of trainable pattern-classifying systems.* New York: McGraw-Hill, 1965.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] A. Sacchetti, B. Bachmann, K. Loffel, U. M. Kunzi, and B. Paoli, "Neural networks to solve partial differential equations: A comparison with finite elements," *IEEE Access*, vol. 10, pp. 32271–32279, 2022.

[4] F. Zagklavara, P. Jimack, N. Kapur, O. Querin, and H. Thompson, "Multi-objective optimisation of polymerase chain reaction continuous flow systems," *Biomedical Microdevices*, January 2022.

[5] B. Bochenek and Z. Ustrnul, "Machine learning in weather prediction and climate analyses-applications and perspectives," *ATMOSPHERE*, vol. 13, FEB 2022.

[6] S. L. Brunton, "Applying machine learning to study fluid mechanics," *Acta Mechanica Sinica/Lixue Xuebao*, vol. 37, pp. 1718–1726, 12 2021.

[7] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2 2019.

[8] S. Cuomo, V. S. D. Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physicsâ€"informed neural networks: Where

we are and whatâ€™s next," *Journal of Scientific Computing*, vol. 92, p. 88, 9 2022.

[9] S. Kollmannsberger, D. D'Angella, M. Jokeit, and L. Herrmann, *Physics-Informed Neural Networks*, pp. 55–84. Cham: Springer International Publishing, 2021.

[10] Q. Zhu, Z. Liu, and J. Yan, "Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks," *Computational Mechanics*, vol. 67, pp. 619–635, 2021.

[11] L. Sun, H. Gao, S. Pan, and J. X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, 4 2020.

[12] M. A. Nabian, R. J. Gladstone, and H. Meidani, "Efficient training of physics-informed neural networks via importance sampling," *Computer-Aided Civil and Infrastructure Engineering*, vol. 36, pp. 962–977, apr 2021.

[13] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 403, 1 2023.

[14] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," 1 2018.

[15] J. Yu, L. Lu, X. Meng, and G. E. Karniadakis, "Gradient-enhanced physics-informed neural networks for forward and inverse pde problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114823, 2022.

[16] W. Wu, X. Feng, and H. Xu, "Improved deep neural networks with domain decomposition in solving partial differential equations," *Journal of Scientific Computing*, vol. 93, 10 2022.

[17] E. Weinan and B. Yu, "The deep ritz method: A deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, pp. 1–12, 3 2018.