Physics-informed neural networks for the data-driven discovery of nonlinear partial differential equations

Kealan J. Hennessy
Department of Civil Engineering and Engineering Mechanics
Columbia University in the City of New York
New York, NY 10027
Email: kjh2182@columbia.edu

***Abstract*—We present the data-driven discovery of a partial differential equation (PDE), namely the one-dimensional continuous time Burger's equation. This discovery is carried out within the framework of a physics-informed neural network, or PINN, which is formulated as a supervised learning task with constraints naturally enforced by the PDE. In the context of Burger's equation, this task involves both the approximation of an unknown solution u, and the learning of two constant parameters λ_1 and λ_2 which define the underlying dynamics of the PDE. We show that the PINN framework is sufficient for carrying out this task when defined via activation functions of sufficient smoothness such as hyperbolic tangent, or `tanh`. We then modify this implementation and instead define the PINN using non-smooth/piecewise linear activation functions (i.e., Rectified Linear Unit, or `ReLU`) in order to showcase the inability of such functions to facilitate learning of the PDE. We offer a rigorous mathematical explanation for why this is the case as a caution for future work.**

I. Introduction

In recent years, the exponential growth in computing power has accelerated the revolution in machine learning to new heights, particularly in the fields of image recognition [2], machine translation [3], cognitive science [4] and genomics [5]. A problem often arises, however, when it comes to analyzing complex physical, biological and engineering systems, and that problem is the cost-prohibitive nature of data acquisition. In many of the scientific endeavours which involve both machine learning and the aforementioned systems, we are forced into a "small data" regime, in which the vast majority of state-of-the-art machine learning techniques (e.g., deep feed-forward/convolutional/recurrent neural networks) lack any robustness, and fail to provide any guarantees of convergence.

Fortunately, there exists an impressive amount of prior knowledge that remains untapped in modern machine learning practice; that is, the vast wealth of human scientific knowledge as it pertains to mathematically formulated laws lies dormant at the time of writing. For the most part, these laws (the principled physical laws that govern the time-dependent dynamics of a system, or some empirically validated rules or other domain expertise, for example), constitute untapped potential when it comes to constraining the space of permissible solutions for a supervised deep learning algorithm to a more practical size. Introducing this constraint properly involves encoding structured information into a learning algorithm, in turn enhancing the informational value of the data that the algorithm *does* see, and thereby enabling it to converge quickly and efficiently towards the correct solution. Proper constraints allow the aforementioned algorithm, when trained on noisy and incomplete data, to generalize with a high level of accuracy. A deep neural network capable of meeting this criteria is henceforth referred to as a *physics-informed deep neural network*, or PINN.[1]

The first objective of this study is to demonstrate a relatively simple yet powerful example of how encoding this information can be done, with a particular emphasis on dealing with (i.e., learning or discovering) the nonlinear dynamics of physical systems. This will be undertaken by replicating (as a base case) a particular forward problem from a recent seminal paper in the field [6] that addresses the issue of representing nonlinear behaviour by leveraging the well-known capability of deep neural networks as universal function approximators [21]. This mission is particularly noteworthy, as thus far in the literature the exploitation of structured prior information through other methods has been met with significant limitations. Perhaps the most common example of these methods is the use of Gaussian process regression (GPR) [10], in which functional representations tailored to a given linear operator are devised, facilitating the accurate inference of solutions to (and uncertainty estimates for) several prototypical problems in mathematical physics. Despite the demonstrable flexibility and mathematical elegance of GPR in encoding prior information, however, issues have arisen when dealing with nonlinearity. For example, in [7], [8] the authors were forced to locally linearize any nonlinear terms in time, therefore limiting the applicability of the proposed GPR method to discrete-time domains, and compromising the accuracy of their predictions for highly nonlinear systems. Issues also arose originating

[1]Once a PINN is trained, its inference capabilities allow it to function in place of traditional solvers which utilize numerical methods to solve PDE. It is important to note, however, that in practice this tradeoff is not definitive; developing PINN's to compete and/or form hybrids with numerical solvers to advance the current state-of-the-art is an ongoing area of research.

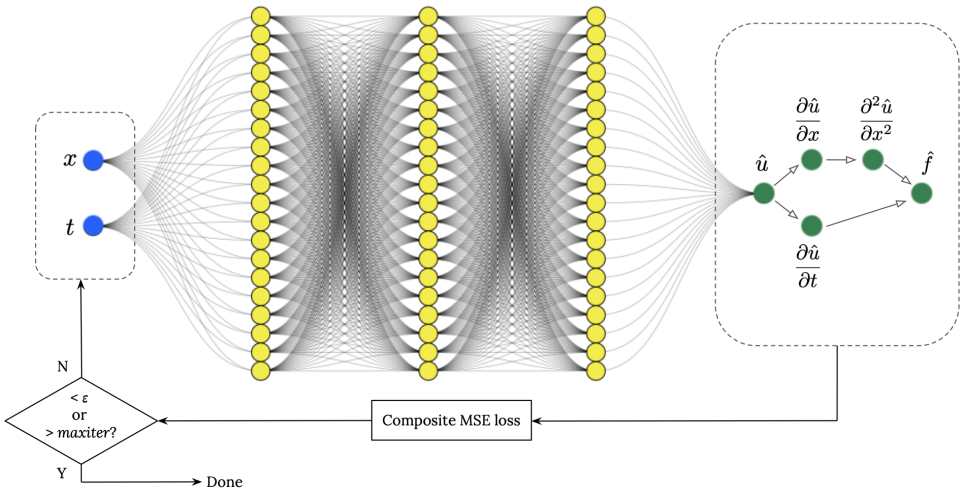

Fig. 1. An illustration of the PINN for the continuous time one-dimensional Burger's equation. The green dots represent the residual network (or, perhaps more accurately, the physics-informed neural network, whose input $\hat{u}$ is the output of the surrogate DNN), while the yellow dots represent the hidden layers of the surrogate DNN, which are preceded by an input layer consisting of the two blue dots (x, t). Note that the size of the surrogate network is arbitrary (for the purpose of visualization, that is), but in this case is 3 layers of 20 units each.

from the Bayesian nature of GPR, which requires certain prior assumptions that fundamentally limit the representation capacity of the model, giving rise to robustness/brittleness issues for nonlinear problems [9].

The secondary goal of this project is to take the replicated form of the original problem and modify it in a deliberate manner so as to further investigate the behaviour of PINN's. This secondary goal will be accomplished by a) observing any noticeable performance differences after this change is made, and if there are any, b) attempting to characterize these differences, and discern exactly why they occur. This modification will be made with careful consideration of common practice from the field of deep learning, so as to "blend" this knowledge with the emerging field of PINN's. The goal here is to elucidate any fundamental differences that occur in practical applications, and if possible, provide the necessary theoretical support.

II. Summary of the Original Paper

A. Methodology of the original paper

The original paper organizes its focus on learning canonical problems in mathematical physics into two different categories, relating both by the general partial differential equation of the form

$$u_t + \mathcal{N}[u; \lambda] = 0, x \in \Omega, t \in [0, T] \tag{1}$$

where $u(t, x)$ denotes the latent (hidden) solution, $\mathcal{N}[; \lambda]$ is a nonlinear operator parametrized by λ, and is a subset of $\mathbb{R}^D$. Defined in this general form, Equation (1) represents a wide range of problems in mathematical physics [11], including conservation laws, advection–diffusion–reaction systems, and diffusion processes, and is the basis of each problem category. The first problem is that of inference, filtering and smoothing - it is the *data-driven solution* of partial differential equations, and is defined by the following question: given limited, noisy measurements of the system, and fixed model parameters λ, what can be said about the unknown hidden state $u(t, x)$. of the system? The second problem, also utilizing limited, noisy measurements, is that of learning, system identification - it is the *data-driven discovery* of partial differential equations. The question it asks is: what are the parameters λ that best describe the observed data? Lastly, note that both of these problem categories are divided into two sub-categories in order to distinguish *continuous time* from *discrete time* modeling. The original paper evaluates the performance and limitations of PINN's for each of these categories and sub-categories; each category is represented by canonical special cases of Equation (1).

Note that PINN learning in general is carried out in a similar manner across all cases, and in a very familiar way - that is, through automatic differentiation [22]. Throughout the original paper, the exact same automatic differentiation techniques employed by the deep learning community are applied to facilitate PINN learning; perhaps more uniquely, the inputs of the PINN (which derivatives are taken with respect to), are the space and time coordinates at which physical phenomena are described by a partial differential equation (one that is well-posed, with a unique solution). The empirical observations of the original paper suggest that this approach introduces a kind of regularization mechanism (the essence of "constraining" the neural network, as previously discussed), which affords the use of relatively simple feed-forward neural network architectures trained with small amounts of data.

B. Key results of the original paper

As discussed, the original paper [6] conducts a systematic review of the capability of physics-informed neural networks to learn the behaviour of various complex nonlinear PDE

through a series of case studies. As these studies vary in their complexity, with most requiring dedicated, powerful hardware (and, in one case, proprietary software) to generate data, only one of them is selected to meet the two objectives of this project.[2] The PDE/case study in question is that of the one-dimensional continuous time Burger's equation, a fundamental convection–diffusion equation that serves as the simplest model for analyzing the combined effect of nonlinear advection and diffusion. This study is defined by four independent variables, namely, the number of layers l and units n of the surrogate network, the introduction of uncorrelated noise into the scattered training data, and the number of collocation points N_u randomly drawn from the training set and fed to the surrogate DNN. The study is conducted by varying the two of four independent variables, while holding the remaining two constant - namely, altering l and n with constant N_u and zero noise, and altering N_u and noise levels while l and n remain constant.

The key results of the original paper are quite straightforward; the methodology proposed therein (see Section II.A for a general overview, and Section III below for more specific details) is shown to be extremely robust with respect to noise levels in the data, yielding reasonable identification accuracy for noise corruption up to 10%. Furthermore, the accuracy of the PINN predictions greatly outperform the aforementioned competing approaches using Gaussian process regression [8]. Thus, in keeping with the first objective of this study, we expect our recreation of the PINN learning process to produce the same level of robustness.

III. Methodology

A. Objective and technical challenges

The particular alteration proposed to the methodology of the original paper (reflecting the secondary objective of this study) is given by the following: we wish to observe the performance (in terms of accuracy) of the trained PINN under a change in activation function, namely from hyperbolic tangent, or `tanh` (used in the original paper), to Rectified Linear Unit, or `ReLU`. For each of these, performance will be evaluated and subsequently compared within the framework of the original paper - i.e., by conducting a systematic investigation involving the same four independent variables, one for each activation function.

The first (and by far the most complex) of two technical challenges overcome in order to enact this comparison was a necessary conversion of the original open source (MIT licensed) codebase[3] provided for recreation of the results of the original paper from Tensorflow 1.x to Tensorflow 2.x (Keras). Dealing with this transition amounted to re-writing the majority of the codebase from scratch, using the original only as a rough guide. This was compulsory in order to avoid the regular use of outdated and/or deprecated functions. Doing so required considerable knowledge of both versions of Tensorflow, each of which hold fundamental differences; with that said, once a working replica was completed, modifications (such as the interchanging of activation functions and initialization methods) was relatively straightforward.

A significant problem encountered in this migration process was the deprecation of `tf.contrib.opt.ScipyOptimizerInterface` with Tensorflow 2.x, which previously housed the L-BGFS-B optimization algorithm utilized in the original (seminal) paper. Though there exists several options for implementing the L-BFGS-B optimizer in TF 2.x, none of them are straightforward, particularly for use in the PINN framework. For example, there exist Python extensions such as Kormos [12] which act as a wrapper to interface `scipy.optimize.minimize` with Keras models (i.e., `tf.model.Sequential()`). As a drop-in replacement for use with modern TF, such packages are in some sense the simplest solution; however, these packages are open-source and very lightly maintained, proving difficult to integrate into the highly specific PINN framework without throwing obscure and largely unfixable errors.

On the other end of the spectrum, there exist domain-specific languages (DSL) for PINN's. In most cases, these are language frameworks with a Tensorflow or similar backend that allow for an easy implementation (i.e., very few lines of code, and a reasonably high level of customization) of common PINN algorithms. Perhaps the most common PINN DSL when it comes to usage in the physical sciences community is `DeepXDE` [13]. All things considered, this was an appealing option for solving the continuous time one-dimensional Burger's equation in its entirety (including the L-BFGS-B optimization stage), and subsequently carrying out a performance evaluation of different activation functions. However, this option also offered little to no choice to utilize or re-create the original code in a similar form. With `DeepXDE`, though the results would be theoretically identical, the code would look and function completely differently - it would abstract away many of the key technical insights, and effectively strain the link between this study and the original paper.

The chosen solution involved a highly customized implementation of the unconstrained L-BFGS optimizer [23] `tfp.optimizers.lbfgs_minimize` available through Tensorflow's probability library [20], a Python library built on TF 2.x that aims to facilitate the combination of probabilistic models with deep learning on modern hardware. The main problem with this method was its lack of a drop-in option for use with a Keras model; that is, it is not implemented as a subclass of `tf.keras.optimizers.Optimizer`. This necessitated the construction of a wrapper to interface the optimizer with a Keras model (and, by extension, the PINN framework), which itself consisted of two separate tasks. The first was the creation of a function `value_and_gradients_function` that returned both

[2]It is also perhaps more interesting to utilize the simplest, most elegant example to conduct such a study, particularly for a "custom" final course project such as this one, in which results are presented to those who may be unfamiliar with PINN's (or the concept of a partial differential equation in general).

[3]https://github.com/maziarraissi/PINNs/tree/master

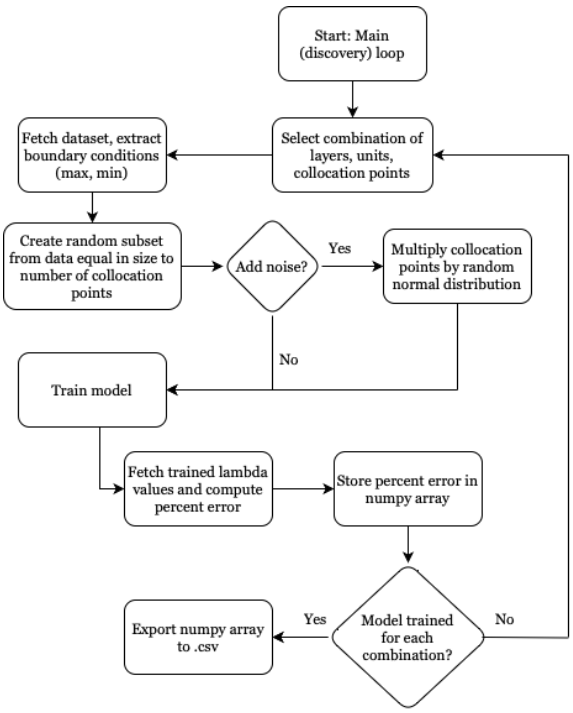

Fig. 2. Flowchart for the main loop that automates the discovery of u, λ_1 and λ_2 via a physics-informed neural network. It is this loop which trains a different model for each combination of the specified independent variables n, l, N_u and noise level.

the gradients and the composite MSE loss as a tuple, a function which `tfp.optimizers.lbfgs_minimize` requires as its input argument. The second was implementing a workaround for the fact that `tfp.optimizers.lbfgs_minimize` expects `value_and_gradients_function` to accept a 1-dimensional `tf.Tensor` as its argument, when in fact TensorFlow and Keras store trainable model parameters as a multi-dimensional collection of `tf.Variable` objects. Furthermore, the output of the L-BFGS-B optimization process likewise produces a 1-dimensional `tf.Tensor`, which must be converted back to a list of multidimensional `tf.Variable` in order to update the parameters of the Keras model. This process of stitching (creating a 1-dimensional tf.Tensor) and partitioning (creating a multi-dimensional parameter space consisting of tf.Variable objects from a 1-dimensional tf.Tensor) became an integral part of model training (see Figure 3).

The second of the two main challenges was the re-working of the weight initialization. Currently, the architecture of the neural networks u and f in the original paper utilize Xavier (or Glorot) initialization [14]. This works perfectly fine for a DNN with `tanh` activation functions; however, as proposed by He et al. [15], using ReLU activation requires a slight modification in order to maximize the positive effects of proper parameter initialization on training, thereby ensuring a scientifically reasonable comparison. Fortunately, in TF 2.x there exist drop-in methods available through `tf.keras.initializers`, which made it quite simple to change the initialization based on the chosen activation function (the latter being an argument passed to the PINN class).

B. Problem formulation and design description

For the problem addressed by this project, we select one of the canonical special cases addressed in the original paper, formulated from the general form of Equation (1) - namely, the data-driven discovery of the one-dimensional continuous time Burgers equation, covered in Appendix B.1 of the original paper. The methodology for this special case (as is as follows. Consider the partial differential equation

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0 \tag{2}$$

where we define

$$f := u_t + \lambda_1 u u_x - \lambda_2 u_{xx} \tag{3}$$

and proceed by first approximating $u(x,t)$ via a deep neural network (DNN). This approximation, called the *surrogate*, is then used to create the resulting physics-informed neural network $f(x,t)$, called the *residual* (see Figure 1).[4] The introduction of this residual network is what encodes the governing physics equations by taking the output of the surrogate and calculating a residual[5] value. The parameters of the surrogate

[4]This is not be confused with Residual networks, or ResNets, the so-called popular neural network architecture whose name is derived from using skip-connections (also known as "residual connections"). This entirely different from calculating the residual of a PDE, as is referred to here.

[5]The term "residual" is borrowed from the field of numerical analysis, and somewhat analogous to the evaluation of a loss function (though it does not function as one in this case). For details, see the discussion in Section V.B.

u and residual f, along with the parameters $\lambda = (\lambda_1, \lambda_2)$ of the differential operator are therefore shared, and learned by minimizing the total mean squared error loss

$$MSE = MSE_u + MSE_f, \tag{4}$$

where

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

and

$$MSE_f = \frac{1}{N}\sum_{i=1}^{N}\left|f\left(t_u^i, x_u^i\right)\right|^2$$

where $\{t_u^i, x_u^i, u^i\}_{i=1}^N$ denote the training data on $u(t, x)$. It is important to note the significance of each component of the loss function. MSE_u represents the square of the difference between the surrogate and the true solution, while MSE_f acts as the portion of the loss function which enforces the structure of the original PDE (which includes both values of λ) at a finite set of collocation points (given by the training data). Together, they complete Equation (4), referred to henceforth as the composite MSE loss.

During the training phase, the surrogate DNN is fed a specified number of discrete data points (x, t) scattered randomly within specified bounds of the spatiotmeporal domain. The optimization methodology of the original paper involves the utilization of two optimizers in succession. The Adam optimizer [16] is utilized first in order to avoid local minima; then, this solution is refined using the second-order constrained Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization scheme [17] to further explore the function terrain and ensure convergence to global minima. Both optimization schemes determine the weights and biases of the surrogate network by minimizing the loss function in Equation (4). Note that the sub-category of the BFGS algorithm most commonly employed for training PINNs is L-BFGS-B. L-BFGS is a limited-memory version of BFGS to handle problems with many variables (i.e., deep learning) and the BFGS-B is the variant of BFGS aimed at addressing bounded constrained optimization problems. Note that throughout the entirety of this study, the stopping condition for Adam and L-BFGS-B differ significantly; while Adam is set to run for a maximum fixed number of iterations (5000, consistent with the original paper) during the training of each model, the L-BFGS-B algorithm is stopped only when the absolute change in the objective value between one iteration and the next is smaller than machine epsilon (which, in the case of 32-bit floating-point operations, is approximately 10^{-16}). Upon training, the calibrated physics-informed neural network facilitates the prediction of the entire solution $u(x, t)$, as well as the parameters $\lambda = (\lambda_1, \lambda_2)$ that define the underlying dynamics.

Note that the only difference between the methodology of the original paper and this study outside of the nuance involved in the aforementioned workarounds/technical challenges (which, admittedly, is expected to contribute to significant discrepancies between the original paper and a replication attempt) is the alteration of the activation function used in the surrogate DNN, tied to the key secondary objective of this study. The overarching methodology (e.g., MSE loss, structure of the PINN algorithm) is identical to the original paper.

IV. Implementation

A. Data

A data set is generated using the exact (analytic) solution for u, and the corresponding exact values of $\lambda_1 = 1.0$ and $\lambda_2 = 0.01/\pi$. Points are generated within the bounded domains $t \in \mathbb{R}, 0 < t \geq 1$ and $x \in \mathbb{R}, -1 \geq x \geq 1$, with a spatial resolution of 256 points, and a temporal resolution of 100 time steps, for a total of 25600 (x, t) tuples. The dataset can be found in the Github repository housing the original codebase, and in the authors Github repository housing the codebase for this study.

B. Deep Learning network

For a visual overview of the architecture of the PINN for the one-dimensional continuous time Burger's equation, see Figure 1. A flowchart of the algorithmic design for the systematic investigation of this PINN's capacity for data-driven discovery (i.e., its ability to learn λ_1 and λ_2) for various configurations of the chosen independent variable pair (either including noise level/number of collocation points N_u, or number of units/number of layers) can be found in Figure 2.

Aside from these figures, there are a few important things to note about the training algorithm. Although the number of collocation points N_u randomly chosen for training is indeed a small subset of the complete dataset (and, as discussed, an independent variable for the present study), for each model both L-BFGS-B and Adam perform optimization using the entirety of that subset in a single-batch fashion; in other words, once the collocation points are randomly chosen to train a given model, that subset of points is not segmented further (batched) during the learning process. Furthermore, note that the input layer for the surrogate network is customized in order to scale the inputs in each forward pass via the relation

$$X = 2\left(\frac{X - \hat{X}_{min}}{\hat{X}_{max} - \hat{X}_{min}}\right) - 1 \tag{5}$$

where $\hat{X}$ is an array of shape $(25600, 2)$ containing the entirety of the dataset, and X is the (varied in size) subset of that data randomly chosen to train the PINN. This scaling is undertaken in order to implicitly enforce boundary and initial conditions.

C. Software design

A flowchart detailing the PINN training process can be found in Figure 3. The complete code (with extensive comments) and requisite data for this study can be found at the authors Github: https://github.com/kealanhennessy/pinns.

TABLE I

PERCENTAGE ERROR IN THE IDENTIFIED PARAMETERS λ_1 AND λ_2, VARYING THE NUMBER OF COLLOCATION POINTS N_u AND NOISE LEVEL, USING THE TANH ACTIVATION FUNCTION. HERE, THE NEURAL NETWORK ARCHITECTURE IS KEPT FIXED TO 9 LAYERS WITH 20 NEURONS PER LAYER.

	% error in λ_1				% error in λ_2			
Noise / N_u	0%	1%	5%	10%	0%	1%	5%	10%
500	0.085	0.102	1.128	0.216	11.195	0.566	2.456	25.478
1000	0.029	0.047	0.464	0.868	0.704	0.144	1.788	6.800
1500	0.059	0.029	0.635	0.015	1.103	0.829	2.183	6.265
2000	0.006	0.105	0.890	0.551	0.173	0.689	4.487	4.877

V. RESULTS

A. Project Results

The main project results are showcased in Tables I-IV. These tables show, for models utilizing either a `tanh` or `ReLU` activation function, the percent accuracy of the error for the discovered values of λ under various changes in the four independent variables utilized in the original study. Training the set of models that utilize a `tanh` activation function took just over four hours; training the set of models utilizing a `ReLU` took approximately one hour). After a fixed number of Adam iterations, the `ReLU` models all converged within less than 500 L-BFGS-B iterations. This is in stark contrast to the optimization of models utilizing `tanh` activation functions, for which the number of L-BFGS-B iterations was close to 8000.

B. Comparison of results to the original paper

To facilitate comparison between this study and the results of the original paper, Tables V and VI display a copy of the the percent error of the predicted values of each λ from the original paper, presented in an identical manner to Tables I and III. It is reasonable to conclude from this close comparison that the first objective of this study has been well met; the accuracy is comparable in most, if not all cases, and any subtle variation can be reasonably attributed to the difference in the implementation of the `PhysicsInformedNN` class utilized for the current study. Most importantly, we observe the same variability and non-monotonic trends in our data (Tables I and III) that appear in the original data (Tables V and VI), behaviour which appears to follow a change in the network architecture and the number collocation points N_u. This behaviour is a key indicator of success in replication of the original problem. Such variability may be attributed to a multitude of different factors pertaining to the behaviour of Burger's equation itself, as well as the particular neural network setup - with that said, a definitive causal link has yet to be drawn between any of these factors and the following results. This presents a series of important questions as the

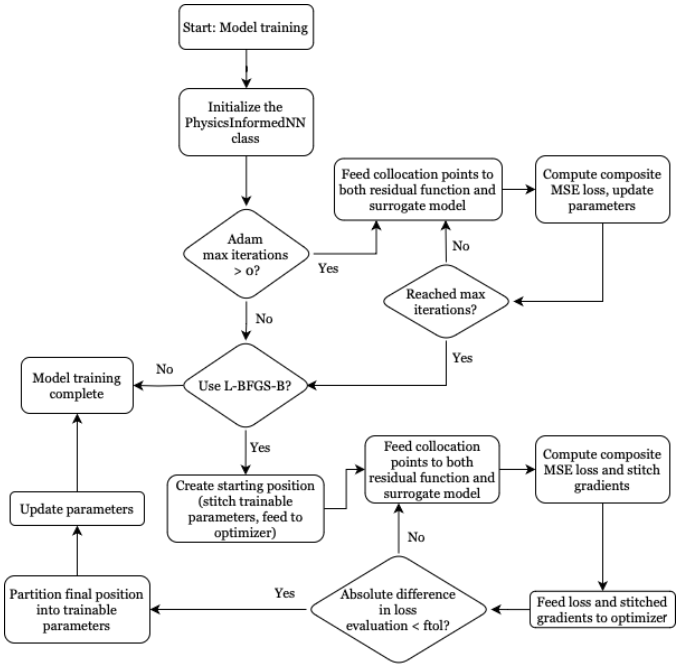

Fig. 3.

TABLE II

PERCENTAGE ERROR IN THE IDENTIFIED PARAMETERS λ_1 AND λ_2, VARYING THE NUMBER OF COLLOCATION POINTS N_u AND NOISE LEVEL, USING THE RELU ACTIVATION FUNCTION. HERE, THE NEURAL NETWORK ARCHITECTURE IS KEPT FIXED TO 9 LAYERS WITH 20 NEURONS PER LAYER.

	% error in λ_1				% error in λ_2			
Noise / N_u	0%	1%	5%	10%	0%	1%	5%	10%
500	94.279	98.336	96.964	94.891	22.128	22.128	22.128	22.128
1000	93.219	96.541	93.337	97.123	22.128	22.128	22.128	22.128
1500	97.381	98.979	96.344	96.323	22.128	22.128	22.128	22.128
2000	97.886	96.996	95.524	97.862	22.128	22.128	22.128	22.128

TABLE III

PERCENTAGE ERROR IN THE IDENTIFIED PARAMETERS λ_1 AND λ_2, VARYING THE NUMBER OF HIDDEN LAYERS l AND NEURONS PER LAYER n, USING THE TANH ACTIVATION FUNCTION. HERE, THE TRAINING DATA IS CONSIDERED TO BE NOISE-FREE AND FIXED TO $N = 2,000$.

	% error in λ_1			% error in λ_2		
Units / Layers	10	20	40	10	20	40
2	29.896	18.328	24.828	107.581	16.358	34.267
4	1.402	0.239	0.156	12.127	2.268	3.718
6	0.049	0.112	0.052	0.710	0.078	0.042
8	0.343	0.025	0.041	1.772	0.329	0.598

TABLE IV

PERCENTAGE ERROR IN THE IDENTIFIED PARAMETERS λ_1 AND λ_2, VARYING THE NUMBER OF HIDDEN LAYERS l AND NEURONS PER LAYER n, USING THE RELU ACTIVATION FUNCTION. HERE, THE TRAINING DATA IS CONSIDERED TO BE NOISE-FREE AND FIXED TO $N = 2,000$.

	% error in λ_1			% error in λ_2		
Units / Layers	10	20	40	10	20	40
2	95.275	95.778	94.085	22.128	22.128	22.128
4	99.597	98.142	98.460	22.128	22.128	22.128
6	98.704	98.898	98.230	22.128	22.128	22.128
8	99.274	98.062	96.967	22.128	22.128	22.128

basis for future research (see Section VI).

C. Discussion

It is abundantly clear from these data that not only is the `tanh` function far superior in terms of accuracy, but furthermore that a PINN using a `ReLU` activation function as the nonlinear component of its surrogate DNN is in fact *unable* to learn the underlying dynamics of a nonlinear partial differential equation. The question remains, of course, as to why this is the case.

The prevailing theory presented by the author is as follows. In order to carry out the optimization of a neural network using PDE-based constraints, the activation function of said network must necessarily possess higher-order derivatives. In other words, PINNs requires sufficiently smooth activation functions in order to function properly; although extraordinarily common in deep learning literature, because `ReLU` is by no means a sufficiently smooth function, its inclusion in the PINN framework does not constitute a consistent (and therefore convergent) solution method. Thus, even in the limit of an infinite training dataset, a PINN utilizing `ReLU`-like activation functions cannot converge to the exact solution of the PDE.

To elaborate, consider the following proposition [18]. Following the seminal work by Goodfellow et al. [19], we can represent a standard feed-forward neural network via a composition of functions, which are themselves either affine-linear maps between units (in successive layers), or scalar nonlinear activations within units, resulting in the representation

$$\mathbf{u}_\theta(y) = C_K \circ \sigma \circ C_{K-1} \ldots\ldots\ldots \circ \sigma \circ C_2 \circ \sigma \circ C_1(y) \quad (6)$$

where the neural network u_θ depends on the tuning of parameters $\theta \in \Theta$, which include weights and biases. Within the standard paradigm of deep learning, one trains the network by fine-tuning θ such that the loss (or effective "distance" in the n-dimensional parameter space) between the neural network and the underlying target is minimized. In the case of PINN's, we wish to find the tuning parameters θ such that the resulting neural network u_θ approximates the unknown solution u of the

TABLE V

REPLICATED FROM THE ORIGINAL PAPER: THE PERCENTAGE ERROR IN THE IDENTIFIED PARAMETERS λ_1 AND λ_2, VARYING THE NUMBER OF COLLOCATION POINTS N_u AND NOISE LEVEL, USING THE TANH ACTIVATION FUNCTION. HERE, THE NEURAL NETWORK ARCHITECTURE IS KEPT FIXED TO 9 LAYERS WITH 20 NEURONS PER LAYER. COMPARE TO TABLE I.

	% error in λ_1				% error in λ_2			
Noise / N_u	0%	1%	5%	10%	0%	1%	5%	10%
500	0.131	0.518	0.118	1.319	13.885	0.483	1.708	4.058
1000	0.186	0.533	0.157	1.869	3.719	8.262	3.481	14.544
1500	0.432	0.033	0.706	0.725	3.093	1.423	0.502	3.156
2000	0.096	0.039	0.190	0.101	0.469	0.008	6.216	6.391

TABLE VI

REPLICATED FROM THE ORIGINAL PAPER: THE PERCENTAGE ERROR IN THE IDENTIFIED PARAMETERS λ_1 AND λ_2, VARYING THE NUMBER OF HIDDEN LAYERS l AND NEURONS PER LAYER n, USING THE TANH ACTIVATION FUNCTION. HERE, THE TRAINING DATA IS CONSIDERED TO BE NOISE-FREE AND FIXED TO $N = 2,000$. COMPARE TO TABLE III.

	% error in λ_1			% error in λ_2		
Units / Layers	10	20	40	10	20	40
2	11.696	2.837	1.679	103.919	67.055	49.186
4	0.332	0.109	0.428	4.721	1.234	6.170
6	0.668	0.629	0.118	3.144	3.123	1.158
8	0.414	0.141	0.266	8.459	1.902	1.552

partial differential equation. In order to do so, we form the basis of the PINN framework by defining the PDE residual

$$\mathcal{R} = \mathcal{R}_\theta := \mathcal{D}\left(\mathbf{u}_\theta\right) - \mathbf{f} \tag{7}$$

where $\mathcal{R} \in Y^*$, and $\|\mathcal{R}\|_{Y^*} < +\infty$ for all $\theta \in \Theta$. It is important to note the similarity between this and the residual of Equation (3), which dictates the structure of the PINN. We proceed by defining the strategy for forward problems that solve for u as the minimization of this residual *simulatenously* over the admissible set of tuning parameters, i.e.

$$\text{Find } \theta^* \in \Theta: \quad \theta^* = \arg\min_{\theta\in\Theta} \left(\|\mathcal{R}_\theta\|_Y\right) \tag{8}$$

which is a generalization of the problem addressed by the PINN framework, proposed by the original paper (and, by extension, this study). To obtain the desired insight, we re-formulate Equation (8) in practical terms as equivalent to the minimization of a functional over the function space Y defined by the set of discrete quadrature points $\mathbb{D}$ and a Lebesgue norm $L^{p_y}(\mathbb{D})$. That is,

$$\text{Find } \theta^* \in \Theta: \quad \theta^* = \arg\min_{\theta\in\Theta} \left(\int_{\mathbb{D}} |\mathcal{R}_\theta(y)|^{p_y} \ \mathrm{d}y\right) \tag{9}$$

where $Y = L^{p_y}(\mathbb{D})$ (i.e., the functions in Y are those for which the p_y-th power of the norm over $\mathbb{D}$ is finite), and dy denotes the d-dimensional Lebesgue measure. This is a common solution for obtaining approximations to nonlinear PDE, and draws inspiration from the field of numerical analysis. As it is not possible to evaluate the integral in Equation (9) exactly, it must be approximated numerically by a quadrature rule. Such a rule can be formulated abstractly by considering the mapping $\mathcal{R}_\theta : \mathbb{D} \mapsto \mathbb{R}^m$. Then, we may approximate said integral via

$$\bar{\mathcal{R}}_N(y) := \sum_{i=1}^{N} w_i \mathcal{R}_\theta\left(y_i\right) \tag{10}$$

which is bounded

$$\left|\bar{\mathcal{R}}_\theta - \bar{\mathcal{R}}_N\right| \leqslant C_q\left(\|\mathcal{R}_\theta\|_{\underline{Y}}, \bar{d}\right) N^{-\alpha}$$

for some $\alpha > 0$, where $\mathcal{R}_\theta \in \underline{Y} \subset Y^* \subset Y$. Here, we have specified both the quadrature points $y_i \in \mathbb{D}$, where $1 \leqslant i \leqslant N$ for some $N \in \mathbb{N}$, and weights w_i with $w_i \in \mathbb{R}_+$. Perhaps most importantly, note that in order for the approximation of Equation (10) to exist (i.e., in order for its error to be sufficiently bounded, with assurances of convergence and stability), $\mathcal{R}_\theta$ must be sufficiently regular.

The key insight here is that the approximation of $\mathcal{R}_\theta$ via quadrature is in fact a general case of the approximation of u carried out by the PINN algorithm - that is, the two share the same mathematical foundations. In fact, if we attempt to re-write Equation (10) using a finite number of quadrature points $\mathcal{S} = \{y_i\}$ with $y_i \in \mathbb{D}$ for all $1 \leqslant i \leqslant N$ (which are analogous to the training data fed to the PINN), we arrive at the *loss function*

$$J(\theta) := \sum_{i=1}^{N} w_i \left|\mathcal{R}_\theta\left(y_i\right)\right|^{p_y} \tag{11}$$

which is analogous to the loss function of Equation (4).

In consideration of the previous analogies, it follows that the class of functions utilized in the approximation of the unknown solution u by the PINN must (as with $\mathcal{R}_\theta$) be

sufficiently regular such that the residual of Equation (3) (as with Equation (7)) may be approximated to a high degree of accuracy, facilitating the minimization of the loss function of Equation (11), that is

$$\theta^* = \arg\min_{\theta \in \Theta}(J(\theta)) \tag{12}$$

which is carried out by the PINN via the L-BFGS-B and Adam optimization schemes. We conclude the following: the regularity condition for PINN's can only be enforced by activation functions that are *sufficiently smooth*, thereby excluding the use of `ReLU` and similar activation functions from the learning process.

VI. Future Work

The work carried out by this study sits comfortably within the rapidly growing body of literature surrounding PINNs. It is quite useful as a theoretical and experimental basis for guiding future scientific investigations concerning the effective training of PINNs (although, it is perhaps most informative in regards to what *not* to do when training PINNs).

As far as the author is concerned, future work connecting directly to this study would certainly involve the exploration of different activation functions for PINN, and the evaluation of their performance with regards to several different classes of nonlinear PDE. Though it may not have been clear from the beginning of this study, it is evident that the field of physics-informed machine learning (at least as it pertains to the approximation of nonlinear PDE) draws heavy influence from the field of numerical analysis. Therefore, as this study attempts to do, future work must endeavour to maintain the link between these two fields, working at their intersection in order to achieve together what the two fields separately cannot.

VII. Conclusion

The data-driven discovery of the continuous time one-dimensional Burger's equation has been carried out via a physics-informed neural network (PINN's), a class of universal function approximators capable of representing the underlying physical laws governing a meager and noisy data-set. This discovery utilized the sufficiently smooth activation function `tanh` to great effect (meeting the first objective of this study), and while the utilization of the `ReLU` proved unsuccessful in the same endeavour, this was shown to be consistent with results from numerical analysis and the theory of deep learning. Furthermore, as a clear modification of the replicated methodology of the original paper, this proof of inefficacy meets the secondary objective of this study. Emphasis is placed on the reasons behind the failure of what is perhaps the most commonly used activation function in the field of deep learning, namely its inability to approximate the unknown solution of a nonlinear partial differential equation due to insufficient smoothness (irregularity).

To wit, as the field of deep learning continues to grow incredibly quickly (both in terms of methodological and algorithmic developments), it is critical that studies such as this, which inform practitioners on the proper use of any given aspect of the PINN framework, be undertaken; after all, there are many applications which can readily benefit from the kind of conclusion offered by this study, including (but certainly not limited to) the data-driven forecasting of physical processes, model predictive control, multi-physics/multi-scale modeling and simulation. This is especially pertinent when it comes to those who may be more accustomed to the field of deep learning, as such practitioners may be tempted to apply popular deep learning methods to develop novel physics-informed neural networks, which (if the current study has shown one thing, and one thing only) require a more sophisticated understanding of the underlying phenomena.

Acknowledgment

The author would like to thank the staff of ECBM E4040 for the effort they have put in to run such an excellent course, and for endeavouring to help every student succeed. Their efforts will live on in the students they have taught and encouraged the past few months.

References

[1] https://github.com/kealanhennessy/pinns/

[2] A. Krizhevsky, I. Sutskever, and G.E. Hinton. *Imagenet classification with deep convolutional neural networks*. Advances in Neural Information Processing Systems, pp. 1097–1105 (2012).

[3] Y. Le Cun, Y. Bengio, and G. Hinton. *Deep learning*. nature 521, pp. 436-444 (2015).

[4] B.M. Lake, R. Salakhutdinov, and J.B. Tenenbaum. *Human-level concept learning through probabilistic program induction*. Science 350, pp. 1332–1338 (2015) .

[5] B. Alipanahi, A. Delong, M.T. Weirauch, and B. J. Frey. *Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning*. Nat. Biotechnol. 33, pp. 831–838 (2015).

[6] 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 378, pp. 686-707 (2019)

[7] M. Raissi, P. Perdikaris, and G.E. Karniadakis. *Numerical Gaussian processes for time-dependent and non-linear partial differential equations*. arXiv: 1703.10230, (2017).

[8] M. Raissi and G.E. Karniadakis. *Hidden physics models: machine learning of nonlinear partial differential equations*. arXiv:1708.00588, (2017).

[9] H. Owhadi, C. Scovel, T. Sullivan, et al. *Brittleness of Bayesian inference under finite information in a continuous world*. Electron. J. Stat. 9, pp. 1–79 (2015).

[10] C.E. Rasmussen and C.K. Williams. Gaussian Processes for Machine Learning, vol. 1. MIT Press, Cambridge (2006).

[11] L. C. Evans, Partial differential equations, vol. 19. American Mathematical Society (2022).

[12] https://github.com/mbhynes/kormos

[13] L. Lu, X. Meng, Z. Mao and G. E. Karniadakis. *DeepXDE: A deep learning library for solving differential equations*. SIAM Review 63 (1), pp. 208-228 (2021).

[14] X. Glorot and Y. Bengio. *Understanding the difficulty of training deep feedforward neural networks*. Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings (2010).

[15] K. He, et al. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*. Proceedings of the IEEE international conference on computer vision (2015).

[16] D P. Kingma and J. Ba. *Adam: A method for stochastic optimization*. arXiv:1412.6980 (2014).

[17] R. Fletcher. Practical methods of optimization. John Wiley & Sons (2000).

[18] S. Mishra and R. Molinaro. *Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs*. IMA Journal of Numerical Analysis 42.2, pp. 981-1022 (2022).

[19] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT Press, Cambridge (2016).
[20] https://www.tensorflow.org/probability
[21] K. Hornik, M. Stinchcombe and H. White. *Multilayer feedforward networks are universal approximators*. Neural Netw. 2, pp. 359–366 (1989).
[22] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, and J.M. Siskind. *Automatic differentiation in machine learning: a survey*. arXiv:1502.05767 (2015).
[23] https://gist.github.com/piyueh/712ec7d4540489aad2dcfb80f9a54993