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

Maziar Raissi[1], Paris Perdikaris[2], and George Em Karniadakis[1]

[1]*Division of Applied Mathematics, Brown University,*
Providence, RI, 02912, USA
[2]*Department of Mechanical Engineering and Applied Mechanics,*
University of Pennsylvania,
Philadelphia, PA, 19104, USA

Abstract

We introduce *physics informed neural networks* – neural networks that are trained to solve supervised learning tasks while respecting any given law of physics described by general nonlinear partial differential equations. In this two part treatise, we present our developments in the context of solving two main classes of problems: data-driven solution and data-driven discovery of partial differential equations. Depending on the nature and arrangement of the available data, we devise two distinct classes of algorithms, namely continuous time and discrete time models. The resulting neural networks form a new class of *data-efficient* universal function approximators that naturally encode any underlying physical laws as prior information. In this first part, we demonstrate how these networks can be used to infer solutions to partial differential equations, and obtain physics-informed surrogate models that are fully differentiable with respect to all input coordinates and free parameters.

Keywords: Data-driven scientific computing, Machine learning, Predictive modeling, Runge-Kutta methods, Nonlinear dynamics

1. Introduction

With the explosive growth of available data and computing resources, recent advances in machine learning and data analytics have yielded transformative results across diverse scientific disciplines, including image recognition [1], natural language processing [2], cognitive science [3], and genomics [4].

However, more often than not, in the course of analyzing complex physical, biological or engineering systems, the cost of data acquisition is prohibitive, and we are inevitably faced with the challenge of drawing conclusions and making decisions under partial information. In this *small data* regime, the vast majority of state-of-the art machine learning techniques (e.g., deep/convolutional/recurrent neural networks) are lacking robustness and fail to provide any guarantees of convergence.

At first sight, the task of training a deep learning algorithm to accurately identify a nonlinear map from a few – potentially very high-dimensional – input and output data pairs seems at best naive. Coming to our rescue, for many cases pertaining to the modeling of physical and biological systems, there a exist a vast amount of prior knowledge that is currently not being utilized in modern machine learning practice. Let it be the principled physical laws that govern the time-dependent dynamics of a system, or some empirical validated rules or other domain expertise, this prior information can act as a regularization agent that constrains the space of admissible solutions to a manageable size (for e.g., in incompressible fluid dynamics problems by discarding any non realistic flow solutions that violate the conservation of mass principle). In return, encoding such structured information into a learning algorithm results in amplifying the information content of the data that the algorithm sees, enabling it to quickly steer itself towards the right solution and generalize well even when only a few training examples are available.

The first glimpses of promise for exploiting structured prior information to construct data-efficient and physics-informed learning machines have already been showcased in the recent studies of [5, 6, 7]. There, the authors employed Gaussian process regression [8] to devise functional representations that are tailored to a given linear operator, and were able to accurately infer solutions and provide uncertainty estimates for several prototype problems in mathematical physics. Extensions to nonlinear problems were proposed in subsequent studies by Raissi *et. al.* [9, 10] in the context of both inference and systems identification. Despite the flexibility and mathematical elegance of Gaussian processes in encoding prior information, the treatment of nonlinear problems introduces two important limitations. First, in [9, 10] the authors had to locally linearize any nonlinear terms in time, thus limiting the applicability of the proposed methods to discrete-time domains and compromising the accuracy of their predictions in strongly nonlinear regimes.

Secondly, the Bayesian nature of Gaussian process regression requires certain prior assumptions that may limit the representation capacity of the model and give rise to robustness/brittleness issues, especially for nonlinear problems [11].

1.1. Problem setup and summary of contributions

In this work we take a different approach by employing deep neural networks and leverage their well known capability as universal function approximators [12]. In this setting, we can directly tackle nonlinear problems without the need for committing to any prior assumptions, linearization, or local time-stepping. We exploit recent developments in automatic differentiation [13] – one of the most useful but perhaps underused techniques in scientific computing – to differentiate neural networks with respect to their input coordinates and model parameters to obtain *physics informed neural networks*. Such neural networks are constrained to respect any symmetry, invariance, or conservation principles originating from the physical laws that govern the observed data, as modeled by general time-dependent and nonlinear partial differential equations. This simple yet powerful construction allows us to tackle a wide range of problems in computational science and introduces a potentially disruptive technology leading to the development of new data-efficient and physics-informed learning machines, new classes of numerical solvers for partial differential equations, as well as new data-driven approaches for model inversion and systems identification.

The general aim of this work is to set the foundations for a new paradigm in modeling and computation that enriches deep learning with the long-standing developments in mathematical physics. These developments are presented in the context of two main problem classes: data-driven solution and data-driven discovery of partial differential equations. To this end, let us consider parametrized and nonlinear partial differential equations of the general form

$$u_t + \mathcal{N}[u; \lambda] = 0,$$

where $u(t, x)$ denotes the latent (hidden) solution and $\mathcal{N}[\cdot; \lambda]$ is a nonlinear operator parametrized by λ. This setup encapsulates a wide range of problems in mathematical physics including conservation laws, diffusion processes, advection-diffusion-reaction systems, and kinetic equations. As a motivating example, the one dimensional Burgers' equation [14] corresponds to the case

where $\mathcal{N}[u; \lambda] = \lambda_1 u u_x - \lambda_2 u_{xx}$ and $\lambda = (\lambda_1, \lambda_2)$. Here, the subscripts denote partial differentiation in either time or space. Given noisy measurements of the system, we are interested in the solution of two distinct problems. The first problem is that of predictive inference, filtering and smoothing, or data driven solutions of partial differential equations [9, 5] which states: given fixed model parameters λ what can be said about the unknown hidden state $u(t, x)$ of the system? The second problem is that of learning, system identification, or data-driven discovery of partial differential equations [10, 6, 15] stating: what are the parameters λ that best describe the observed data?

In this first part of our two-part treatise, we focus on computing data-driven solutions to partial differential equations of the general form

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

where $u(t, x)$ denotes the latent (hidden) solution, $\mathcal{N}[\cdot]$ is a nonlinear differential operator, and Ω is a subset of $\mathbb{R}^D$. In what follows, we put forth two distinct classes of algorithms, namely continuous and discrete time models, and highlight their properties and performance through the lens of different benchmark problems. All code and data-sets accompanying this manuscript are available at https://github.com/maziarraissi/PINNs.

2. Continuous Time Models

We define $f(t, x)$ to be given by the left-hand-side of equation (1); i.e.,

$$f := u_t + \mathcal{N}[u], \tag{2}$$

and proceed by approximating $u(t, x)$ by a deep neural network. This assumption along with equation (2) result in a *physics informed neural network* $f(t, x)$. This network can be derived by applying the chain rule for differentiating compositions of functions using automatic differentiation [13].

2.1. Example (Burgers' Equation)

As an example, let us consider the Burgers' equation. This equation arises in various areas of applied mathematics, including fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow [14]. It is a fundamental partial differential equation and can be derived from the Navier-Stokes equations for the velocity field by dropping the pressure gradient term. For small values of

the viscosity parameters, Burgers' equation can lead to shock formation that is notoriously hard to resolve by classical numerical methods. In one space dimension, the Burger's equation along with Dirichlet boundary conditions reads as

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \qquad (3)$$
$$u(0, x) = -\sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$

Let us define $f(t, x)$ to be given by

$$f := u_t + uu_x - (0.01/\pi)u_{xx},$$

and proceed by approximating $u(t, x)$ by a deep neural network. To highlight the simplicity in implementing this idea we have included a Python code snippet using Tensorflow [16]; currently one of the most popular and well documented open source libraries for machine learning computations. To this end, $u(t, x)$ can be simply defined as

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

Correspondingly, the *physics informed neural network* $f(t, x)$ takes the form

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

The shared parameters between the neural networks $u(t, x)$ and $f(t, x)$ can be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_f, \qquad (4)$$

where

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

and

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

Here, $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(t, x)$ and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ specify the collocations points for $f(t, x)$. The loss MSE_u corresponds to the initial and boundary data while MSE_f enforces the structure imposed by equation (3) at a finite set of collocation points.

In all benchmarks considered in this work, the total number of training data N_u is relatively small (a few hundred up to a few thousand points), and we chose to optimize all loss functions using L-BFGS; a quasi-Newton, full-batch gradient-based optimization algorithm [17]. For larger data-sets a more computationally efficient mini-batch setting can be readily employed using stochastic gradient descent and its modern variants [18, 19]. Despite the fact that there is no theoretical guarantee that this procedure converges to a global minimum, our empirical evidence indicates that, if the given partial differential equation is well-posed and its solution is unique, our method is capable of achieving good prediction accuracy given a sufficiently expressive neural network architecture and a sufficient number of collocation points N_f. This general observation deeply relates to the resulting optimization landscape induced by the mean square error loss of equation 4, and defines an open question for research that is in sync with recent theoretical developments in deep learning [20, 21]. Here, we will test the robustness of the proposed methodology using a series of systematic sensitivity studies that accompany the numerical results presented in the following.

Figure 1 summarizes our results for the data-driven solution of the Burgers equation. Specifically, given a set of $N_u = 100$ randomly distributed initial and boundary data, we learn the latent solution $u(t, x)$ by training all 3021 parameters of a 9-layer deep neural network using the mean squared error loss of (4). Each hidden layer contained 20 neurons and a hyperbolic tangent activation function. In general, the neural network should be given sufficient approximation capacity in order to accommodate the anticipated

complexity of $u(t, x)$. However, in this example, our choice aims to highlight the robustness of the proposed method with respect to the well known issue of over-fitting. Specifically, the term in MSE_f in equation (4) acts as a regularization mechanism that penalizes solutions that do not satisfy equation (3). Therefore, a key property of *physics informed neural networks* is that they can be effectively trained using small data sets; a setting often encountered in the study of physical systems for which the cost of data acquisition may be prohibitive.

The top panel of Figure 1 shows the predicted spatio-temporal solution $u(t, x)$, along with the locations of the initial and boundary training data. We must underline that, unlike any classical numerical method for solving partial differential equations, this prediction is obtained without any sort of discretization of the spatio-temporal domain. The exact solution for this problem is analytically available [14], and the resulting prediction error is measured at $6.7 \cdot 10^{-4}$ in the relative $\mathcal{L}_2$-norm. Note that this error is about two orders of magnitude lower than the one reported in our previous work on data-driven solution of partial differential equation using Gaussian processes [9]. A more detailed assessment of the predicted solution is presented in the bottom panel of figure 1. In particular, we present a comparison between the exact and the predicted solutions at different time instants $t = 0.25, 0.50, 0.75$. Using only a handful of initial and boundary data, the *physics informed neural network* can accurately capture the intricate nonlinear behavior of the Burgers' equation that leads to the development of a sharp internal layer around $t = 0.4$. The latter is notoriously hard to accurately resolve with classical numerical methods and requires a laborious spatio-temporal discretization of equation (3).

To further analyze the performance of our method, we have performed the following systematic studies to quantify its predictive accuracy for different number of training and collocation points, as well as for different neural network architectures. In table 1 we report the resulting relative $\mathcal{L}_2$ error for different number of initial and boundary training data N_u and different number of collocation points N_f, while keeping the 9-layer network architecture fixed. The general trend shows increased prediction accuracy as the total number of training data N_u is increased, given a sufficient number of collocation points N_f. This observation highlights a key strength of *physics informed neural networks*: by encoding the structure of the underlying phys-
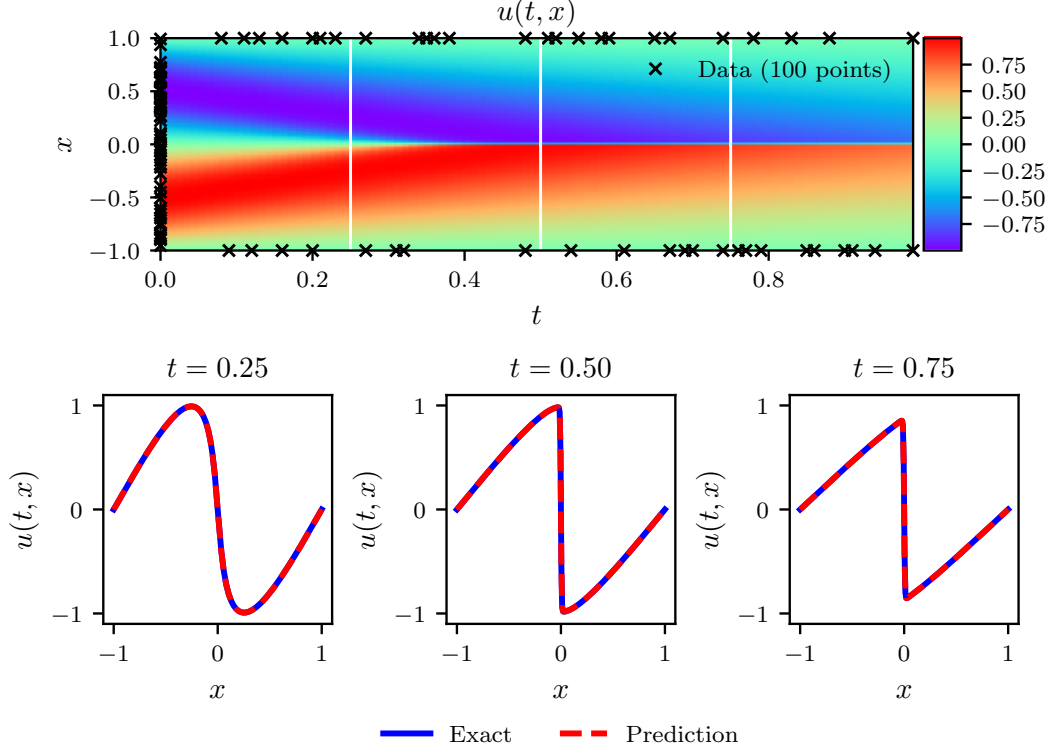
Figure 1: *Burgers' equation: Top:* Predicted solution $u(t, x)$ along with the initial and boundary training data. In addition we are using 10,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the white vertical lines in the top panel. The relative $\mathcal{L}_2$ error for this case is $6.7 \cdot 10^{-4}$. Model training took approximately 60 seconds on a single NVIDIA Titan X GPU card.

ical law through the collocation points N_f, one can obtain a more accurate and data-efficient learning algorithm.[1] Finally, table 2 shows the resulting relative $\mathcal{L}_2$ for different number of hidden layers, and different number of neurons per layer, while the total number of training and collocation points is kept fixed to $N_u = 100$ and $N_f = 10,000$, respectively. As expected, we observe that as the number of layers and neurons is increased (hence the capacity of the neural network to approximate more complex functions), the

---

[1] Note that the case $N_f = 0$ corresponds to a standard neural network model, i.e., a neural network that does not take into account the underlying governing equation.

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

Table 1: *Burgers' equation:* Relative $\mathcal{L}_2$ error between the predicted and the exact solution $u(t, x)$ for different number of initial and boundary training data N_u, and different number of collocation points N_f. Here, the network architecture is fixed to 9 layers with 20 neurons per hidden layer.

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

Table 2: *Burgers' equation:* Relative $\mathcal{L}_2$ error between the predicted and the exact solution $u(t, x)$ for different number of hidden layers and different number of neurons per layer. Here, the total number of training and collocation points is fixed to $N_u = 100$ and $N_f = 10,000$, respectively.

predictive accuracy is increased.

2.2. Example (Shrödinger Equation)

This example aims to highlight the ability of our method to handle periodic boundary conditions, complex-valued solutions, as well as different types of nonlinearities in the governing partial differential equations. The one-dimensional nonlinear Schrödinger equation is a classical field equation that is used to study quantum mechanical systems, including nonlinear wave propagation in optical fibers and/or waveguides, Bose-Einstein condensates, and plasma waves. In optics, the nonlinear term arises from the intensity dependent index of refraction of a given material. Similarly, the nonlinear term for Bose-Einstein condensates is a result of the mean-field interactions of an interacting, N-body system. The nonlinear Schrödinger equation along

9

with periodic boundary conditions is given by

$$ih_t + 0.5h_{xx} + |h|^2h = 0, \quad x \in [-5, 5], \quad t \in [0, \pi/2], \qquad (5)$$
$$h(0, x) = 2\,\text{sech}(x),$$
$$h(t, -5) = h(t, 5),$$
$$h_x(t, -5) = h_x(t, 5),$$

where $h(t, x)$ is the complex-valued solution. Let us define $f(t, x)$ to be given by

$$f := ih_t + 0.5h_{xx} + |h|^2h,$$

and proceed by placing a complex-valued neural network prior on $h(t, x)$. In fact, if u denotes the real part of h and v is the imaginary part, we are placing a multi-out neural network prior on $h(t, x) = \begin{bmatrix} u(t, x) & v(t, x) \end{bmatrix}$. This will result in the complex-valued (multi-output) *physic informed neural network* $f(t, x)$. The shared parameters of the neural networks $h(t, x)$ and $f(t, x)$ can be learned by minimizing the mean squared error loss

$$MSE = MSE_0 + MSE_b + MSE_f, \qquad (6)$$

where

$$MSE_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |h(0, x_0^i) - h_0^i|^2,$$

$$MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(|h^i(t_b^i, -5) - h^i(t_b^i, 5)|^2 + |h_x^i(t_b^i, -5) - h_x^i(t_b^i, 5)|^2 \right),$$

and

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

Here, $\{x_0^i, h_0^i\}_{i=1}^{N_0}$ denotes the initial data, $\{t_b^i\}_{i=1}^{N_b}$ corresponds to the collocation points on the boundary, and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ represents the collocation points on $f(t, x)$. Consequently, MSE_0 corresponds to the loss on the initial data, MSE_b enforces the periodic boundary conditions, and MSE_f penalizes the Schrödinger equation not being satisfied on the collocation points.

In order to assess the accuracy of our method, we have simulated equation

([5](#)) using conventional spectral methods to create a high-resolution data set. Specifically, starting from an initial state $h(0, x) = 2\,\text{sech}(x)$ and assuming periodic boundary conditions $h(t, -5) = h(t, 5)$ and $h_x(t, -5) = h_x(t, 5)$, we have integrated equation ([5](#)) up to a final time $t = \pi/2$ using the Chebfun package [22] with a spectral Fourier discretization with 256 modes and a fourth-order explicit Runge-Kutta temporal integrator with time-step $\Delta t = \pi/2 \cdot 10^{-6}$. Under our data-driven setting, all we observe are measurements $\{x_0^i, h_0^i\}_{i=1}^{N_0}$ of the latent function $h(t, x)$ at time $t = 0$. In particular, the training set consists of a total of $N_0 = 50$ data points on $h(0, x)$ randomly parsed from the full high-resolution data-set, as well as $N_b = 50$ randomly sampled collocation points $\{t_b^i\}_{i=1}^{N_b}$ for enforcing the periodic boundaries. Moreover, we have assumed $N_f = 20,000$ randomly sampled collocation points used to enforce equation ([5](#)) inside the solution domain. All randomly sampled point locations were generated using a space filling Latin Hypercube Sampling strategy [23].

Here our goal is to infer the entire spatio-temporal solution $h(t, x)$ of the Schrödinger equation ([5](#)). We chose to jointly represent the latent function $h(t, x) = [u(t, x)\ v(t, x)]$ using a 5-layer deep neural network with 100 neurons per layer and a hyperbolic tangent activation function. Figure [2](#) summarizes the results of our experiment. Specifically, the top panel of figure [2](#) shows the magnitude of the predicted spatio-temporal solution $|h(t, x)| = \sqrt{u^2(t, x) + v^2(t, x)}$, along with the locations of the initial and boundary training data. The resulting prediction error is validated against the test data for this problem, and is measured at $1.97 \cdot 10^{-3}$ in the relative $\mathcal{L}_2$-norm. A more detailed assessment of the predicted solution is presented in the bottom panel of Figure [2](#). In particular, we present a comparison between the exact and the predicted solutions at different time instants $t = 0.59, 0.79, 0.98$. Using only a handful of initial data, the *physics informed neural network* can accurately capture the intricate nonlinear behavior of the Schrödinger equation.

One potential limitation of the continuous time neural network models considered so far, stems from the need to use a large number of collocation points N_f in order to enforce physics informed constraints in the entire spatio-temporal domain. Although this poses no significant issues for problems in one or two spatial dimensions, it may introduce a severe bottleneck in higher dimensional problems, as the total number of collocation points needed
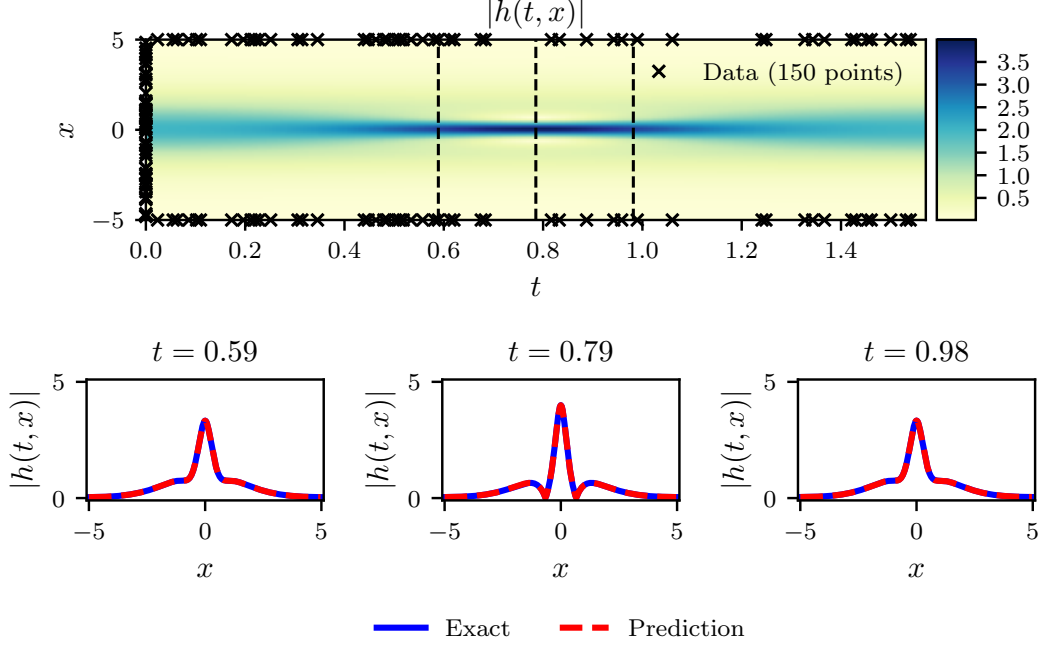
Figure 2: *Shrödinger equation: Top:* Predicted solution $|h(t, x)|$ along with the initial and boundary training data. In addition we are using 20,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the dashed vertical lines in the top panel. The relative $\mathcal{L}_2$ error for this case is $1.97 \cdot 10^{-3}$.

to globally enforce a physics informed constrain (i.e., in our case a partial differential equation) will increase exponentially. In the next section, we put forth a different approach that circumvents the need for collocation points by introducing a more structured neural network representation leveraging the classical Runge-Kutta time-stepping schemes [24].

3. Discrete Time Models

Let us apply the general form of Runge-Kutta methods with q stages [24] to equation (1) and obtain

$$
\begin{aligned}
u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, \ldots, q, \\
u^{n+1} &= u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}].
\end{aligned}
\tag{7}
$$

12

Here, $u^{n+c_j}(x) = u(t^n + c_j \Delta t, x)$ for $j = 1, \ldots, q$. This general form encapsulates both implicit and explicit time-stepping schemes, depending on the choice of the parameters $\{a_{ij}, b_j, c_j\}$. Equations (7) can be equivalently expressed as

$$u^n = u^n_i, \quad i = 1, \ldots, q,$$
$$u^n = u^n_{q+1}, \tag{8}$$

where

$$u^n_i := u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, \ldots, q,$$
$$u^n_{q+1} := u^{n+1} + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}]. \tag{9}$$

We proceed by placing a multi-output neural network prior on

$$\left[u^{n+c_1}(x), \ldots, u^{n+c_q}(x), u^{n+1}(x) \right]. \tag{10}$$

This prior assumption along with equations (9) result in a *physics informed neural network* that takes x as an input and outputs

$$\left[u^n_1(x), \ldots, u^n_q(x), u^n_{q+1}(x) \right]. \tag{11}$$

3.1. Example (Burgers' Equation)

To highlight the key features of the discrete time representation we revisit the problem of data-driven solution of the Burgers' equation. For this case, the nonlinear operator in equation (9) is given by

$$\mathcal{N}[u^{n+c_j}] = u^{n+c_j} u^{n+c_j}_x - (0.01/\pi) u^{n+c_j}_{xx},$$

and the shared parameters of the neural networks (10) and (11) can be learned by minimizing the sum of squared errors

$$SSE = SSE_n + SSE_b, \tag{12}$$

where

$$SSE_n = \sum_{j=1}^{q+1} \sum_{i=1}^{N_n} |u^n_j(x^{n,i}) - u^{n,i}|^2,$$

13

and

$$SSE_b = \sum_{i=1}^{q} \left(|u^{n+c_i}(-1)|^2 + |u^{n+c_i}(1)|^2 \right) + |u^{n+1}(-1)|^2 + |u^{n+1}(1)|^2.$$

Here, $\{x^{n,i}, u^{n,i}\}_{i=1}^{N_n}$ corresponds to the data at time t^n. The Runge-Kutta scheme now allows us to infer the latent solution $u(t, x)$ in a sequential fashion. Starting from initial data $\{x^{n,i}, u^{n,i}\}_{i=1}^{N_n}$ at time t^n and data at the domain boundaries $x = -1$ and $x = 1$, we can use the aforementioned loss function (12) to train the networks of (10), (11), and predict the solution at time t^{n+1}. A Runge-Kutta time-stepping scheme would then use this prediction as initial data for the next step and proceed to train again and predict $u(t^{n+2}, x)$, $u(t^{n+3}, x)$, etc., one step at a time.

In classical numerical analysis, these steps are usually confined to be small due to stability constraints for explicit schemes or computational complexity constrains for implicit formulations [24]. These constraints become more severe as the total number of Runge-Kutta stages q is increased, and, for most problems of practical interest, one needs to take thousands to millions of such steps until the solution is resolved up to a desired final time. In sharp contrast to classical methods, here we can employ implicit Runge-Kutta schemes with an arbitrarily large number of stages at effectively no extra cost.[2] This enables us to take very large time steps while retaining stability and high predictive accuracy, therefore allowing us to resolve the entire spatio-temporal solution in a single step.

The result of applying this process to the Burgers' equation is presented in figure 3. For illustration purposes, we start with a set of $N_n = 250$ initial data at $t = 0.1$, and employ a *physics informed neural network* induced by an implicit Runge-Kutta scheme with 500 stages to predict the solution at time $t = 0.9$ in a single step. The theoretical error estimates for this scheme predict a temporal error accumulation of $\mathcal{O}(\Delta t^{2q})$ [24], which in our case translates into an error way below machine precision, i.e., $\Delta t^{2q} = 0.8^{1000} \approx 10^{-97}$. To our knowledge, this is the first time that an implicit Runge-Kutta scheme

---

[2]To be precise, it is only the number of parameters in the last layer of the neural network that increases linearly with the total number of stages.
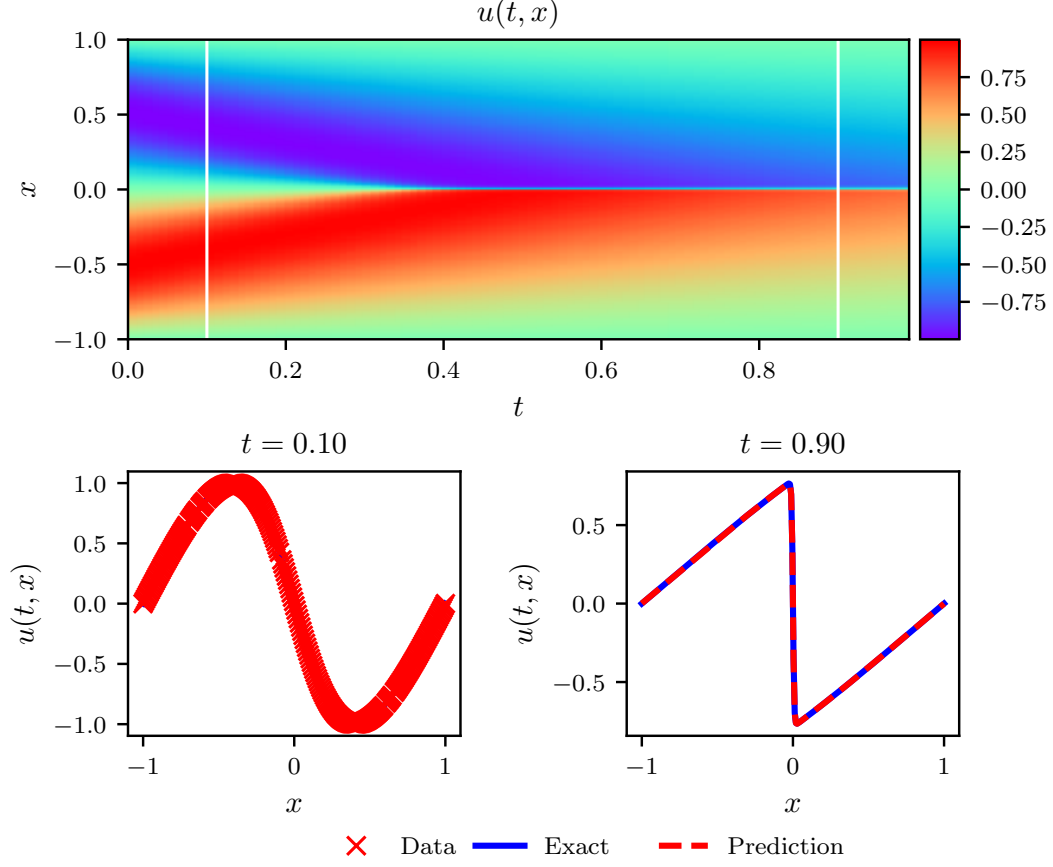
Figure 3: *Burgers equation: Top:* Solution $u(t, x)$ along with the location of the initial training snapshot at $t = 0.1$ and the final prediction snapshot at $t = 0.9$. *Bottom:* Initial training data and final prediction at the snapshots depicted by the white vertical lines in the top panel. The relative $\mathcal{L}_2$ error for this case is $8.2 \cdot 10^{-4}$.

of that high-order has ever been used. Remarkably, starting from smooth initial data at $t = 0.1$ we can predict the nearly discontinuous solution at $t = 0.9$ in a single time-step with a relative $\mathcal{L}_2$ error of $8.2 \cdot 10^{-4}$. This error is two orders of magnitude lower that the one reported in [9], and it is entirely attributed to the neural network's capacity to approximate $u(t, x)$, as well as to the degree that the sum of squared errors loss allows interpolation of the training data. The network architecture used here consists of 4 layers with 50 neurons in each hidden layer.

Neurons / Layers	10	25	50
1	4.1e-02	4.1e-02	1.5e-01
2	2.7e-03	5.0e-03	2.4e-03
3	3.6e-03	1.9e-03	9.5e-04

Table 3: *Burgers' equation:* Relative final prediction error measure in the $\mathcal{L}_2$ norm for different number of hidden layers and neurons in each layer. Here, the number of Runge-Kutta stages is fixed to 500 and the time-step size to $\Delta t = 0.8$.

A detailed systematic study to quantify the effect of different network architectures is presented in table 3. By keeping the number of Runge-Kutta stages fixed to $q = 500$ and the time-step size to $\Delta t = 0.8$, we have varied the number of hidden layers and the number of neurons per layer, and monitored the resulting relative $\mathcal{L}_2$ error for the predicted solution at time $t = 0.9$. Evidently, as the neural network capacity is increased the predictive accuracy is enhanced.

The key parameters controlling the performance of our discrete time algorithm are the total number of Runge-Kutta stages q and the time-step size Δt. In table 4 we summarize the results of an extensive systematic study where we fix the network architecture to 4 hidden layers with 50 neurons per layer, and vary the number of Runge-Kutta stages q and the time-step size Δt. Specifically, we see how cases with low numbers of stages fail to yield accurate results when the time-step size is large. For instance, the case $q = 1$ corresponding to the classical trapezoidal rule, and the case $q = 2$ corresponding to the 4th-order Gauss-Legendre method, cannot retain their predictive accuracy for time-steps larger than 0.2, thus mandating a solution strategy with multiple time-steps of small size. On the other hand, the ability to push the number of Runge-Kutta stages to 32 and even higher allows us to take very large time steps, and effectively resolve the solution in a single step without sacrificing the accuracy of our predictions. Moreover, numerical stability is not sacrificed either as implicit Runge-Kutta is the only family of time-stepping schemes that remain A-stable regardless of their order, thus making them ideal for stiff problems [24]. These properties are unprecedented for an algorithm of such implementation simplicity, and illustrate one of the key highlights of our discrete time approach.

Δt q	0.2	0.4	0.6	0.8
1	3.5e-02	1.1e-01	2.3e-01	3.8e-01
2	5.4e-03	5.1e-02	9.3e-02	2.2e-01
4	1.2e-03	1.5e-02	3.6e-02	5.4e-02
8	6.7e-04	1.8e-03	8.7e-03	5.8e-02
16	5.1e-04	7.6e-02	8.4e-04	1.1e-03
32	7.4e-04	5.2e-04	4.2e-04	7.0e-04
64	4.5e-04	4.8e-04	1.2e-03	7.8e-04
100	5.1e-04	5.7e-04	1.8e-02	1.2e-03
500	4.1e-04	3.8e-04	4.2e-04	8.2e-04

Table 4: *Burgers' equation:* Relative final prediction error measured in the $\mathcal{L}_2$ norm for different number of Runge-Kutta stages q and time-step sizes Δt. Here, the network architecture is fixed to 4 hidden layers with 50 neurons in each layer.

3.1.1. Example (Allen-Cahn Equation)

This example aims to highlight the ability of the proposed discrete time models to handle different types of nonlinearity in the governing partial differential equation. To this end, let us consider the Allen-Cahn equation along with periodic boundary conditions

$$u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \qquad (13)$$
$$u(0, x) = x^2 \cos(\pi x),$$
$$u(t, -1) = u(t, 1),$$
$$u_x(t, -1) = u_x(t, 1).$$

The Allen-Cahn equation is a well-known equation from the area of reaction-diffusion systems. It describes the process of phase separation in multi-component alloy systems, including order-disorder transitions. For the Allen-Cahn equation, the nonlinear operator in equation (9) is given by

$$\mathcal{N}[u^{n+c_j}] = -0.0001u_{xx}^{n+c_j} + 5\left(u^{n+c_j}\right)^3 - 5u^{n+c_j},$$

and the shared parameters of the neural networks (10) and (11) can be learned by minimizing the sum of squared errors

$$SSE = SSE_n + SSE_b, \qquad (14)$$

17

where

$$SSE_n = \sum_{j=1}^{q+1} \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2,$$

and

$$
\begin{aligned}
SSE_b &= \sum_{i=1}^{q} |u^{n+c_i}(-1) - u^{n+c_i}(1)|^2 + |u^{n+1}(-1) - u^{n+1}(1)|^2 \\
&+ \sum_{i=1}^{q} |u_x^{n+c_i}(-1) - u_x^{n+c_i}(1)|^2 + |u_x^{n+1}(-1) - u_x^{n+1}(1)|^2.
\end{aligned}
$$

Here, $\{x^{n,i}, u^{n,i}\}_{i=1}^{N_n}$ corresponds to the data at time t^n. We have generated a training and test data-set set by simulating the Allen-Cahn equation (13) using conventional spectral methods. Specifically, starting from an initial condition $u(0,x) = x^2 \cos(\pi x)$ and assuming periodic boundary conditions $u(t,-1) = u(t,1)$ and $u_x(t,-1) = u_x(t,1)$, we have integrated equation (13) up to a final time $t = 1.0$ using the Chebfun package [22] with a spectral Fourier discretization with 512 modes and a fourth-order explicit Runge-Kutta temporal integrator with time-step $\Delta t = 10^{-5}$.

In this example, we assume $N_n = 200$ initial data points that are randomly sub-sampled from the exact solution at time $t = 0.1$, and our goal is to predict the solution at time $t = 0.9$ using a single time-step with size $\Delta t = 0.8$. To this end, we employ a discrete time *physics informed neural network* with 4 hidden layers and 200 neurons per layer, while the output layer predicts 101 quantities of interest corresponding to the $q = 100$ Runge-Kutta stages $u^{n+c_i}(x)$, $i = 1, \ldots, q$, and the solution at final time $u^{n+1}(x)$. Figure 4 summarizes our predictions after the network has been trained using the loss function of equation (14). Evidently, despite the complex dynamics leading to a solution with two sharp internal layers, we are able to obtain an accurate prediction of the solution at $t = 0.9$ using only a small number of scattered measurements at $t = 0.1$.

4. Summary and Discussion

We have introduced *physics informed neural networks*, a new class of universal function approximators that is capable of encoding any underlying
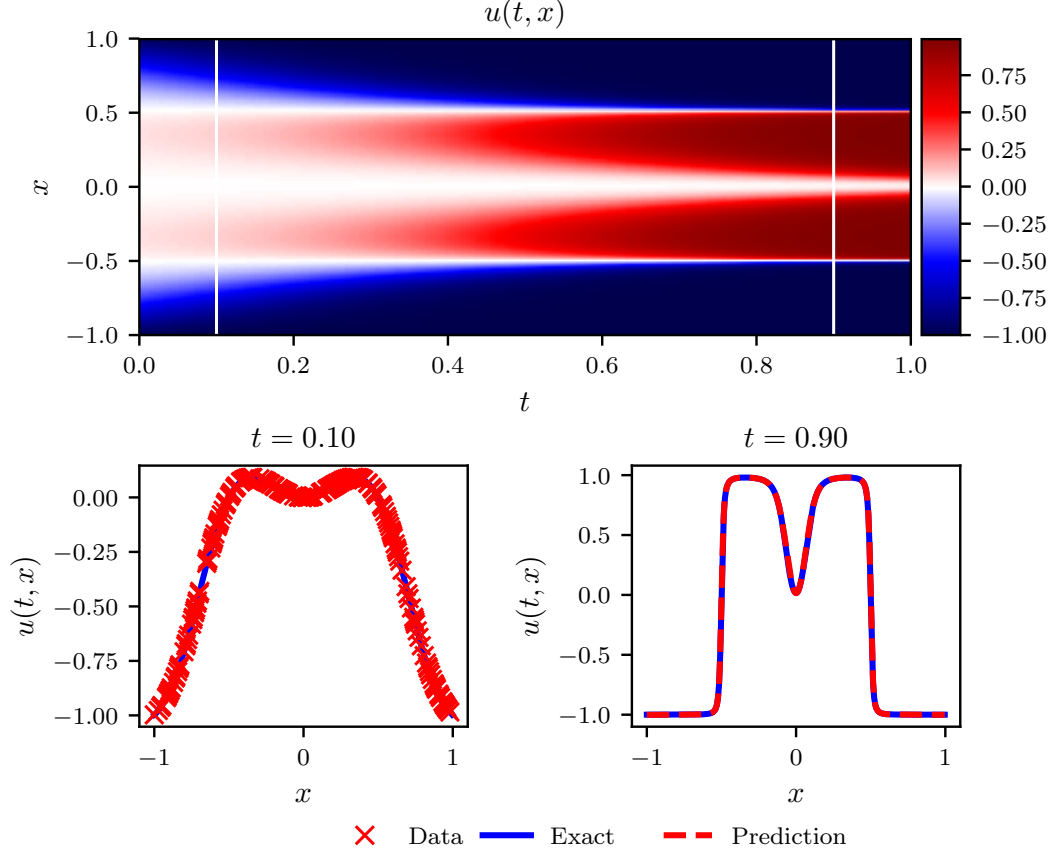
18

Figure 4: *Allen-Cahn equation: Top:* Solution $u(t, x)$ along with the location of the initial training snapshot at $t = 0.1$ and the final prediction snapshot at $t = 0.9$. *Bottom:* Initial training data and final prediction at the snapshots depicted by the white vertical lines in the top panel. The relative $\mathcal{L}_2$ error for this case is $6.99 \cdot 10^{-3}$.

physical laws that govern a given data-set, and can be described by partial differential equations. In this work, we design data-driven algorithms for inferring solutions to general nonlinear partial differential equations, and constructing computationally efficient physics-informed surrogate models. The resulting methods showcase a series of promising results for a diverse collection of problems in computational science, and open the path for endowing deep learning with the powerful capacity of mathematical physics to model the world around us. As deep learning technology is continuing to grow rapidly both in terms of methodological and algorithmic developments, we believe that this is a timely contribution that can benefit practitioners across

a wide range of scientific domains. Specific applications that can readily enjoy these benefits include, but are not limited to, data-driven forecasting of physical processes, model predictive control, multi-physics/multi-scale modeling and simulation.

We must note however that the proposed methods should not be viewed as replacements of classical numerical methods for solving partial differential equations (e.g., finite elements, spectral methods, etc.). Such methods have matured over the last 50 years and, in many cases, meet the robustness and computational efficiency standards required in practice. Our message here, as advocated in Section 3, is that classical methods such as the Runge-Kutta time-stepping schemes can coexist in harmony with deep neural networks, and offer invaluable intuition in constructing structured predictive algorithms. Moreover, the implementation simplicity of the latter greatly favors rapid development and testing of new ideas, potentially opening the path for a new era in data-driven scientific computing. This will be further highlighted in the second part of this paper in which *physics informed neural networks* are put to the test of data-driven discovery of partial differential equations.

Finally, in terms of future work, one pressing question involves addressing the problem of quantifying the uncertainty associated with the neural network predictions. Although this important element was naturally addressed in previous work employing Gaussian processes [9], it not captured by the proposed methodology in its present form and requires further investigation.

Acknowledgements

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

[2] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.

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

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

[5] M. Raissi, P. Perdikaris, G. E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, Journal of Computational Physics 335 (2017) 736–746.

[6] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, Journal of Computational Physics 348 (2017) 683 – 693.

[7] H. Owhadi, Bayesian numerical homogenization, Multiscale Modeling & Simulation 13 (2015) 812–828.

[8] C. E. Rasmussen, C. K. Williams, Gaussian processes for machine learning, volume 1, MIT press Cambridge, 2006.

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

[10] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, arXiv preprint arXiv:1708.00588 (2017).

[11] H. Owhadi, C. Scovel, T. Sullivan, et al., Brittleness of Bayesian inference under finite information in a continuous world, Electronic Journal of Statistics 9 (2015) 1–79.

[12] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (1989) 359–366.

[13] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation in machine learning: a survey, arXiv preprint arXiv:1502.05767 (2015).

[14] C. Basdevant, M. Deville, P. Haldenwang, J. Lacroix, J. Ouazzani, R. Peyret, P. Orlandi, A. Patera, Spectral and finite difference solutions of the Burgers equation, Computers & fluids 14 (1986) 23–41.

[15] S. H. Rudy, S. L. Brunton, J. L. Proctor, J. N. Kutz, Data-driven discovery of partial differential equations, Science Advances 3 (2017).

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv preprint arXiv:1603.04467 (2016).

[17] D. C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Mathematical programming 45 (1989) 503–528.

[18] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.

[19] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[20] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, Y. LeCun, The loss surfaces of multilayer networks, in: Artificial Intelligence and Statistics, pp. 192–204.

[21] R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural networks via information, arXiv preprint arXiv:1703.00810 (2017).

[22] T. A. Driscoll, N. Hale, L. N. Trefethen, Chebfun guide, 2014.

[23] M. Stein, Large sample properties of simulations using latin hypercube sampling, Technometrics 29 (1987) 143–151.

[24] A. Iserles, A first course in the numerical analysis of differential equations, 44, Cambridge University Press, 2009.