Solving Partial Differential Equations with Neural Networks

Juan B. Pedro ¹ Juan Maroñas ² Roberto Paredes ²

Abstract

Many scientific and industrial applications require solving Partial Differential Equations (PDEs) to describe the physical phenomena of interest. Some examples can be found in the fields of aerodynamics, astrodynamics, combustion and many others. In some exceptional cases an analytical solution to the PDEs exists, but in the vast majority of the applications some kind of numerical approximation has to be computed.

In this work, an alternative approach is proposed using neural networks (NNs) as the approximation function for the PDEs. Unlike traditional numerical methods, NNs have the property to be able to approximate any function given enough parameters. Moreover, these solutions are continuous and derivable over the entire domain removing the need for discretization. Another advantage that NNs as function approximations provide is the ability to include the free-parameters in the process of finding the solution. As a result, the solution can generalize to a range of situations instead of a particular case, avoiding the need of performing new calculations every time a parameter is changed dramatically decreasing the optimization time.

We believe that the presented method has the potential to disrupt the physics simulation field enabling real-time physics simulation and geometry optimization without the need of big supercomputers to perform expensive and time consuming simulations.

1. Introduction

Many scientific and industrial applications require solving Partial Differential Equations (PDEs) to describe physical phenomena such as sound, heat, diffusion, electrostatics, electrodynamics, fluid dynamics, elasticity, or quantum mechanics. Some examples of real-world applications can be found in the fields of aerodynamics for aircraft design, neuroscience for brain activity simulation, etc. Hence, the ability to solve PDEs fast and accurately is an active field of research and industrial interest, and the main motivation of this study (Leveque, 1990).

In some exceptional cases, yet useful, an analytic solution to the PDEs exists. Nevertheless, the vast majority of interesting real-world applications require some kind of numerical approximation that has to be computed. The numerical simulation of PDEs has been a topic of interest for the scientific community for a long time. Numerous techniques exist today to solve PDEs. Some examples are finite elements, finite difference, finite volumes and Galerkin methods (Randall & Leveque, 2002) (Toro, 2009) (Pirozzoli, 2011).

All these techniques are based on the idea of domain discretization: divide the computational domain of interest where the PDEs are to be solved and assume a form for the solution on each of these sub-regions. These trail solutions can be as simple as constant values or more elaborated high-order polynomial reconstruction. The global solution is recovered by simply putting together each individual solution. The process of finding the solution to the PDEs consists then on finding the values for the different parameters that minimizes the approximation error. Depending on the numerical technique used, these solutions will provide different properties. Nevertheless, they all share some features directly derived from the discretization process. Namely, they are discontinuous and with limited derivability. Moreover, their accuracy directly depends on the level of precision of the discretization. Numerical schemes used to approximate the solution are constrained by physical effects concerning the rate of change of information between discretized elements. All these aspects usually result in a large amount of discretized elements, which in turn makes some applications unfeasible.

In order to overcome the previously mentioned limitations of traditional techniques, in this work we explore the use of neural networks (NNs) as the solution approximation for PDEs.

¹Sensio AI Solutions ²PRHLT Research Center, Universitat Politècnica de València. Correspondence to: Juan B. Pedro <sensioai@gmail.com>.

2. Related Work

Some data-driven approaches to solve PDEs with NNs exist nowadays. In these cases, a lot of simulations are carried out in order to generate a big amount of data that is then used to train NNs in a supervised manner. This is not the approach that we take, since our objective is to remove expensive simulations out of the picture. Our approach is unsupervised and some works that also follow this idea can be found in the literature.

Chiaramonte and Kiener (Chiaramonte & Kiener) use a NN with a single hidden layer and a trial solution to obtain continuous differentiable solution to PDEs. They use the independent variables and a bias as input parameters, and one single 10 output with the optimal values of the trial solution that satisfies the PDE. They test it in the Laplace equation and a conservation law. The results show relatively small errors when compared to the analytical solutions. The advantage of this method is that the obtained solution is a smooth approximation that can be evaluated and differentiated continuously on the domain. Several areas of improvement include adaptive training set generation to reduce training costs and the study of non-uniform discretizations.

Parisi et al. (Parisi et al., 2003) use an unsupervised approach to train a NN to solve PDEs. They take advantage of the universal approximation capabilities of NNs to postulate them as a solution for a given PDE. A single hidden layer perceptron is used as a generic function. The weights are then found by gradient descent optimisation using the original PDE and a set of sample points as error function, using a genetic algorithm for their initialisation. They compared their solution with a traditional method in an unsteady solidgas reactor problem which relied on spatial discretization obtaining similar accuracy results but at a fraction of the time since once the NN is trained it can find the solution at any given point instantaneously.

Baymani et al. (Baymani et al., 2015) use a NN to solve the Navier Stokes (NS) equations. An analytical solution formed by two parts (one that satisfies boundary conditions and other for the internal domain) is found via optimisation in a feed-forward network with two hidden layers. Results obtained by this method for a two-dimensional steady problem show good agreement with existing data giving smaller errors compared to traditional numerical methods. Furthermore, the solution generated by the NN can be reused at any time.

In (Sirignano & Spiliopoulos, 2017) Sirignano and Spiliopoulos develop an algorithm similar to Galerkin methods in order to approximate high-order PDEs. Their method is mesh-less, and the NN is trained to satisfy the differential operators and boundary conditions using stochastic gradient descent at randomly sampled spatial points. A similar work

is presented in Han et al. (Han et al., 2017) who also studied the use of NNs to approximate high-dimensional PDEs.

During the time of preparation of this paper, NVIDIA presented a real-world application of the method at SC19, the annual supercomputing conference, in Denver (NVIDIA, 2019), min. 43:40. In the conference they present a NN trained to solve the heat flow in a heat sink. By training the NN using the geometry parameters as inputs, they are able to solve the PDEs in a wide range of configurations. They show the real-time heat flow as the geometry changes as well as a new optimal configuration never found before. However, no written work has been found in the literature describing their solution.

Here we summarise the advantages that solving PDEs with NNs present when compared to traditional methods are:

- Continuous and derivable solution over the domain, not piecewise discrete (mesh-less).
- Computational complexity does not increase with the number of sampling points.
- Free parameters can be included in the solution, avoiding repeating simulations at different conditions.
- Once the NN is trained, it can be reused to obtain results instantly.

Although the main disadvantage of this method is the expensive training, the advantages overcome the limitations since once the training is completed, the NN can be reused over and over.

Our main contribution is the use of a single NN to provide the solution for the PDE in the entire domain. This means we are not assuming any a priori solution form or custom functions to satisfy boundary conditions (a pattern present in almost every reviewed work). Another aspect is the use of NNs with more than 2 layers, an architecture not explored in previous works.

3. Background

PDEs are equations that contain unknown multivariate functions and their partial derivatives. This study is restricted to second-order conservation laws of the form

$$\phi_t + \nabla \cdot (\mathbf{u} \cdot \phi) = \nabla \cdot (\Gamma \nabla \phi) \tag{1}$$

Where ϕ is the dependent variable, $\phi_t = \partial \phi/\partial t$ is the derivative of ϕ w.r.t. to time, t, $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$ is the nabla operator, $\mathbf{u} = (u, v, w)$ is the velocity and Γ is the diffusivity. The independent variables are space, $\mathbf{x} = (x, y, z)$, and time, t. Equation 1 is known as the convection-diffusion

equation and it describes physical phenomena where particles, energy, or other physical quantities are transferred inside a physical system due to two processes: diffusion and convection. Concerning diffusion, $\nabla \cdot (\Gamma \nabla \phi)$, one can assume that ϕ is the concentration of a chemical. When concentration is low somewhere compared to the surrounding areas (e.g. a local minimum of concentration), the substance will diffuse in from the surroundings, so the concentration will increase. Conversely, if concentration is high compared to the surroundings (e.g. a local maximum of concentration), then the substance will diffuse out and the concentration will decrease. The net diffusion is proportional to the Laplacian (or second derivative) of concentration if the diffusivity Γ is a constant, $\Gamma \nabla^2 \phi$. On the other hand, concerning convection, $\nabla \cdot (\mathbf{u} \cdot \phi)$, imagine the chemical is being transported through a river and we are measuring the water's concentration each second. Upstream, somebody dumps a bucket of the chemical into the river. A while later, you would see the concentration suddenly rise, then fall, as the zone with increased chemical passes by.

The problems presented in this study as examples will consist on finding the function $\phi(\mathbf{x},t)$ that satisfies the PDE for a given geometry and initial and boundary conditions. In a traditional method like finite volume, we first divide the computational domain in small regions where the volume average value of ϕ at a particular instant is considered

$$\phi_i^n = \frac{1}{V_i} \int_{V_i} \phi(\mathbf{x}, t^n) \tag{2}$$

Where V_i denotes the volume of the ith discretized element in the computational domain. One can then obtain the global solution at any time by gathering the individual solutions for all volumes.

$$\phi^n = \sum_{V_i} \phi_i^n \tag{3}$$

This function is piece-wise constant and is not derivable, showing the first limitations of traditional numerical methods. Having derivable PDE solutions is important for many applications, since some interesting results are computed with the derivatives, i.e. heat fluxes, mass transfer, etc, A derivable solution will be more accurate than derivatives approximation using piece-wise functions.

In order to update the solution in time, we discretize the different operators in equation 1 and use a time integration scheme. In the simplest form, using the first-order Euler algorithm,

$$\phi_i^{n+1} = \phi_i^n - \frac{\Delta t}{V_i} \sum_{f_i} F_f A_f \tag{4}$$

where Δt is the time step, F_f is the flux of ϕ through the face f of the volume V_i and A_f is the area of the face. In order to compute the fluxes at the faces, we require additional numerical schemes to approximate these unknown values. Some popular choices are central difference or upwind methods.

In order to close the problem we need two additional elements: the initial condition and the boundary conditions. First, note that an initial condition is just a boundary condition for the time dimension. This is mentioned because initial and boundary conditions are usually treated separately but in our approach they will be treated equally. The initial condition sets the value for $\phi(\mathbf{x},t=0)$ and in the previously explained time-marching algorithm serves as the first values to start the computation. Boundary conditions, on the other hand, sets the value for $\phi(\mathbf{x} \in D, t>0)$ where D are all the points that lie in the boundary of the domain. Since we cannot see values outside the domain we have to define a particular set of rules to update these points. A lot of boundary conditions exist, but the ones used in our examples are the following:

- 1. Periodic: By assuming periodic conditions the domain folds itself to connect boundaries.
- 2. Dirichlet: For this type of boundary conditions we will fix ϕ at boundaries.
- 3. Newmann: For this type of boundary conditions we will fix $\nabla \phi$ at boundaries.

Special boundaries may be required for the treatment of walls, inflows or outflows. Note that an initial condition is a Dirichlet boundary condition in the temporal dimension.

Other traditional methods like finite difference or finite elements slightly differ on the methodology used, but the same idea underlies: discretize the computational domain in small regions where a form of the solution is assumed and then recover the global solution by putting them all together. This results in piece-wise solutions which are not derivable. Also, since we use time-marching algorithms, new computations are required every time that we change the free-parameters, initial or boundary conditions.

4. Methodology

In this section we present a methodology to find a solution for equation 1 using a neural network. Our goal is to be able to obtain a trained multi-layer perceptron (MLP) that can output $\phi(\mathbf{x},t)$ when \mathbf{x} and t are set as inputs that also satisfies equation 1. The main idea here is that using the independent variables as NN inputs, a forward pass on the network gives us the value of the dependent variables evaluated at that particular point. Since NNs are derivable,

we can compute the derivatives of the dependent variables (outputs) w.r.t. the dependent variables (inputs) in order to compute the different derivatives that appear in the original PDEs. With this derivatives, we build a loss function that matches the PDEs and that is used during the training process. If the loss function reaches a near-zero value, we can assume that our NN is indeed a solution to the PDEs. The training process is unsupervised. These solutions are continuous and derivable over the entire domain. An additional interesting property is that NNs allows us to include the free-parameters of the PDEs as part of the solution. As a result, a solution trained for different values of these parameters can generalize to a range of situations instead of a particular case, avoiding the need of performing new calculations every time a parameter is changed. This property is of particular interest in optimisation studies.

Going into more detail, we define a set of points inside our domain in the same way that we would do in traditional methods. We divide these points into two sets, one for training the NN and the other for validation during training. We also distinguish between internal points and boundary points. These last will be treated accordingly to the specified boundary conditions (see figure 2).

Then, we define the MLP architecture: a number of layers and a number of hidden units in each layer. The number of inputs will be equal to the number of independent variables on the PDEs in addition to any free-parameter that we wish to include. The number of outputs will correspond to the number of unknowns to be solved.

Once we have our training data and the NN defined, the process to find the solution is defined as follows:

- For all points, we compute the outputs of the network, $\phi(\mathbf{x},t)$ and the derivatives w.r.t the inputs: ϕ_t , ϕ_x , ϕ_{xx} , etc.
- For internal points, we use a loss function that matches our PDE. This is the function we want to optimize for: $\phi_t + \nabla \cdot (\mathbf{u} \cdot \phi) \nabla \cdot (\Gamma \nabla \phi) = 0$
- For boundary points, since we fix values, we can build a MSE loss function to satisfy the specified condition.
- Update the parameters of the NN for each loss function.

As it can be seen, the process of training the NN requires the optimization with respect to many loss functions (as many as PDEs and different boundary conditions) which can be challenging in complex problems and the main limitation found by the authors so far.

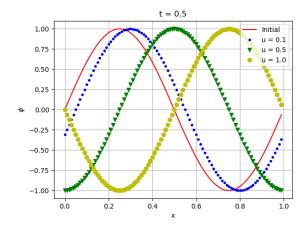


Figure 1. Example of the solution to the one-dimensional advection equation with the initial condition $sin(2\pi x)$

5. Results

In this section we illustrate the methodology presented above with two examples. First, a simple one-dimensional advection equation is solved to understand the main process of solving a PDE with a NN. Then, a more challenging two-dimensional case involving first and second order derivatives is solved to showcase the potential of the method, also introducing free parameters as part of the solution.

5.1. One-dimensional Advection equation

Consider the one-dimensional advection eqaution which is a simplification of equation 1 for a one-dimensional inviscid case

$$\phi_t + u\phi_x = 0 \tag{5}$$

where $\phi(x,t)$ is the unknown function, x and t are the independent variables, u is a constant parameter and ϕ_t and ϕ_x are the derivatives of ϕ w.r.t t and x respectively. This PDE has analytical solution, which is $\phi(x,t) = \phi(x,x-ut)$. From a physical point of view we can say that the initial condition $\phi(x,t=0)$ is moving in x at the speed u. In the case that $\phi(x,t=0) = \sin(2\pi x/L)$ the solution is $\phi(x,t) = \sin(2\pi(x-ut)/L)$ as illustrated in figure 1.

To solve the equation we first define a set of points for training. Defining a Δx and Δt allows us to build a uniform grid of points in the entire domain (see figure 2). We define internal and boundary points, which will have different associated loss functions. In this case, the initial condition will use a Mean Square Error loss function that will compare the initial condition (which is known) with the NN output whenever t=0. For the spatial boundary condition, we set a periodic condition that will also use a Mean Square Error

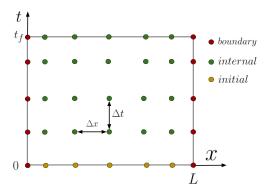


Figure 2. Example of training points with N=5 and M=4

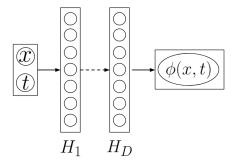


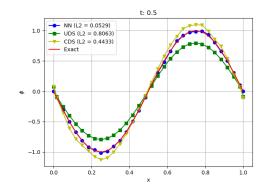
Figure 3. Example of solution.

loss function to compare the solutions at x=0 and x=L for any t and force them to be equal.

Also, we define our solution as a multi-layer perceptron with 2 inputs (number of independent variables), D number of hidden layers and 1 output (number of unknowns) as depicted in figure 3.

The training goes as follows:

- For the internal points, compute the network's outputs: $\phi(0 < x < L, t > 0)$.
- Compute the gradients of the outputs w.r.t the inputs: ϕ_x, ϕ_t .
- Build the loss function for internal points: $L_1 = MSE(\phi_t + u\phi_x)$
- Compute outputs for boundary conditions: $\phi(0 < x < L, t = 0)$, $\phi(x = 0, t)$ and $\phi(x = L, t)$.
- Build the loss function for boundary conditions: $L_2 = MSE(\phi(0 < x < L, t = 0) sin(2\phi x/L), L_3 = MSE(\phi(x = 0, t) \phi(x = L, t))$



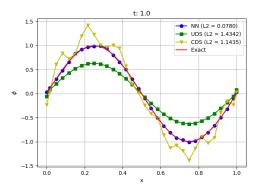


Figure 4. Solution for the one-dimensional advection equation.

• Update the NN parameters for the different losses.

Results for a 5 hidden layer NN with 32 hidden units and u=1 can be seen in figure 4. Results are compared with traditional FVM with an upwind scheme (UDS) and a central scheme (CDS), both time-integrated with an Euler scheme. Unlike the FVM solution, our trained NN is continuous and derivable over the entire domain. Our experiments show that, for this simple case, we can obtain better results with a much bigger mesh since our methodology is not restricted by physical effects.

5.2. Two-dimensional Advection-Diffusion equation

We applied the same methodology introduced in the previous section to solve the viscous Smith-Hutton problem.

$$(u\phi)_x + (v\phi)_y = \Gamma(\phi_{xx} + \phi_{yy}) \tag{6}$$

In this case we are interested in the steady solution of the two-dimensional advection-diffusion equation in the domain depicted in figure 5. The solution used can be seen in figure 6

Results obtained for a 60x30 grid and 3 different values of Γ are shown in figure 7. A 4 layer NN with 1024 hidden units

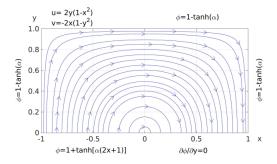


Figure 5. Smith-Hutton problem domain.

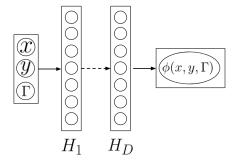


Figure 6. Smith-Hutton solution architecture.

in each layer and Sigmoid activation functions was used. Results are in very good agreements with experimental results. It is important to note that all the solutions were obtained by the same NN. Moreover, it is capable of generalize to other conditions not seen during training with an acceptable level of accuracy.

6. Conclusions

In this work we have presented a methodology to solve PDEs using NNs. Compared to traditional numerical techniques, our approach is able to provide accurate solutions which are continuous and derivable in the entire domain. Furthermore, free-parameters can be included as NN inputs obtaining solutions in a wide range of conditions. This can dramatically decrease the optimization time of problems where the numerical resolution of PDEs is required.

The proposed methodology consists on using the independent variables of the PDEs as NN inputs. A forward pass on the network gives us the value of the dependent variables evaluated at that particular point. Since NNs are derivable, we can compute the derivatives of the dependent variables (outputs) w.r.t. the dependent variables (inputs) in order to compute the different derivatives that appear in the original PDEs. With this derivatives, we can build a loss function

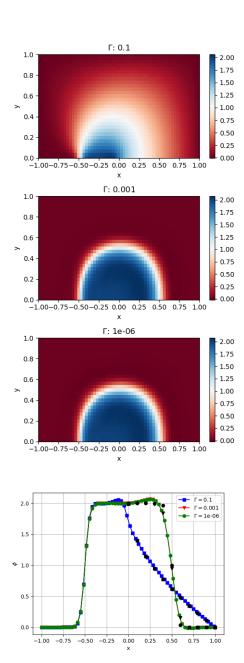


Figure 7. Smith-Hutton solution fields (top) and inlet-outlet profiles (bottom).

that matches the PDEs and that is used during the training process, which is unsupervised.

We tested our method in two cases. First, a simple one-dimensional advection equation is solved showing that with a 5 hidden layer NN and 32 hidden units we can provide more accurate solutions than using traditional finite volume methods. Our second case involves the resolution of a two-dimensional advection-diffusion equation. This PDE involves both first and second order derivatives as well as the diffusive parameter introduced as an additional NN input. This results in a NN that is able to provide a continuous solution to the PDE over the entire computational domain and for a wide range of physical conditions. This result dramatically decreases the optimization process and is eye-opening on the multitude of applications that our approach can impact.

We believe that the presented method has the potential to disrupt the physics simulation field. This powerful technique can be used, for example, to solve the flow over an entire aircraft in a wide range of geometrical and flight conditions. Then it is possible the visualisation of the flow in real time as we change the geometry and physical conditions to obtain the optimal configuration. However, some issues must be addressed before.

When working with systems of PDEs and multiple boundary conditions, our method requires the optimisation of the NN with respect to a lot of loss functions. This results in a big restriction when training the NN. Also, some training strategies can be devised to reach better results, such as mesh refinement or data augmentation tailored to our application.

References

- Baymani, M., Effati, S., Niazmand, H., and Kerayechian., A. Artificial neural network method for solving the navier stokes equations. 26(4):765–763, 2015.
- Chiaramonte, M. M. and Kiener, M. Solving differential equations using neural networks.
- Han, J., Jentzen, A., and E., W. Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning. 2017.
- Leveque, R. *Numerical Methods for Conservation Laws*. Basel: Birkhauser-Verlag, 1990.
- NVIDIA. https://www.youtube.com/watch?v=69nEEpdEJzU, 2019.
- Parisi, D. R., Mariani, M. C., and Laborde., M. A. Solving differential equations with unsupervised neural networks. 42(8-9):715–721, 2003.

- Pirozzoli, S. Tnumerical methods for high-speed flows. *Annu. Rev. Fluid Mech*, 43::16394, 2011.
- Randall, J. and Leveque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.
- Sirignano, J. and Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. 2017.
- Toro, E. F. Riemann Solvers and Numerical Methods for Fluid Dynamics. New York: Springer, 2009.