Proceedings of Machine Learning Research vol 120:1–10, 2020

VarNet: Variational Neural Networks for the Solution of Partial Differential Equations

Reza Khodayi-mehr REZA.KHODAYI.MEHR@DUKE.EDU
Michael M. Zavlanos MICHAEL.ZAVLANOS@DUKE.EDU
Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA.

Abstract

We propose a new model-based unsupervised learning method, called VarNet, for the solution of partial differential equations (PDEs) using deep neural networks. Particularly, we propose a novel loss function that relies on the variational (integral) form of PDEs as opposed to their differential form which is commonly used in the literature. Our loss function is discretization-free, highly parallelizable, and more effective in capturing the solution of PDEs since it employs lower-order derivatives and trains over measure non-zero regions of space-time. The models obtained using VarNet are smooth and do not require interpolation. They are also easily differentiable and can directly be used for control and optimization of PDEs. Finally, VarNet can straight-forwardly incorporate parametric PDE models making it a natural tool for model order reduction of PDEs.

Keywords: Neural networks, deep learning, partial differential equations, dynamical systems, model order reduction, advection-diffusion transport.

1. Introduction

Dynamical systems are typically modeled using differential equations that capture their input-output behavior. Particularly, for spatiotemporal systems the underlying dynamics are modeled using partial differential equations (PDEs) whose states live in infinite-dimensional spaces. Except for some special cases, only approximate solutions to such systems can be obtained using discretization-based numerical methods. These methods typically belong to one of three main categories, namely, finite difference (FD) methods, finite element (FE) methods, and finite volume (FV) methods; see Ames (2014). These methods return solutions defined over a set of grid points that sample the spatiotemporal domain; the solution at any other point can be obtained by interpolating the nearby nodal values. This local treatment of the solution makes the models obtained from these methods extremely expressive but also very expensive to store and evaluate.

In this paper we follow a different approach that relies on deep neural networks (NNs) to capture the solutions of PDEs using trainable parameters that are considerably fewer than the number of grid points used in discretization-based methods. Using NNs to approximate solutions of PDEs can be beneficial for the following reasons: (i) their evaluation is extremely fast and thus, unlike currently available model order reduction (MOR) methods, there is no need to compromise accuracy for speed, (ii) parallelization of the training is trivial, and (iii) the resulting model is smooth and differentiable and thus, it can be readily used in PDE-constrained optimization problems, e.g., for source identification Khodayi-mehr et al. (2019) or control of PDEs Khodayi-mehr and Zavlanos (2020). The authors in (Yadav et al., 2015, Ch. 04) provide a review of different approaches for solving PDEs using NNs. One group of approaches utilize NNs to memorize the solution of

PDEs. Particularly, they solve the PDE using a numerical method to obtain labeled training data and often utilize convolutional NNs (CNNs), being powerful image processing tools, to capture the numerical solution in a supervised learning way Khoo et al. (2017). For instance, Guo et al. (2016) propose a fluid dynamics solver that utilizes a CNN to predict the velocity field in steady-state problems resulting in a stellar speedup of four orders-of-magnitude in computation time. Long et al. (2017) propose PDE-NET that is capable of identifying partial differential operators as well as approximating the solution of PDEs. Note that these approaches do not replace numerical methods but rather rely on them and introduce an extra layer of approximation. There exist another group of methods called FE-NNs which represent the governing FE equations at the element level using artifical neurons Xu et al. (2012); Ramuhalli et al. (2005). These approaches scale with the number of discretization points and are similar in spirit to numerical methods like the FE method.

Most relevant to the approach proposed in this paper is a set of works that also directly train a NN to approximate the solution of PDEs in an unsupervised learning way. One of the early works of this kind is proposed by Lagaris et al. (1998) that uses the residual of the differential form of the PDE to define the required loss function. In order to remove the constraints from the training problem, the authors only consider simple domains for which the boundary conditions (BCs) can be manually enforced by a change of variables. Avrutskiy (2017) elaborates more on this technique. Although these approaches attain comparable accuracy to numerical methods, they are impractical since in general enforcing the BCs is as difficult as solving the original PDE. Following a different approach, Rudd (2013) utilizes a constrained back-propagation algorithm to enforce the initial and boundary conditions during training. In order to avoid solving a constrained training problem, Shirvany et al. (2009) add the constraints corresponding to BCs to the objective as penalty terms. Similarly, Sirignano and Spiliopoulos (2017) focus on the solution of PDEs with high dimensions using a long short-term memory architecture and prove a convergence result for the solution as the number of trainable parameters of the NN is increased. A similar approach is proposed by Raissi et al. (2019) that utilizes the physical laws modeled by PDEs as regularizing terms to guide the learning process, while Zhu et al. (2019) use the PDEs to define energy fields that are minimized to train CNNs to predict the PDE solutions at discrete sets of points. Alternatively, reinforcement learning can be used to train NNs that approximate the solutions of PDEs. For instance, Wei et al. (2018) use the actor-critic algorithm where the PDE residual acts as the critic.

Compared to this literature, the contributions of this paper are as follows: (i) The literature discussed above uses the PDE residual in its differential form as the loss function which requires an extremely large number of training points to adequately learn the PDE solution. Instead, we propose a novel loss function that relies on the variational (integral) form of the PDE. This loss function contains lower order derivatives so that the solution of the PDE can be estimated more accurately. Mroeover, it considers segments of space-time as opposed to single points and imposes fewer smoothness requirements on the solution. (ii) We develop the VARNET library Khodayi-mehr and Zavlanos (2019a) that uses the proposed deep learning framework to solve PDEs. Using NNs to approximate solutions of PDEs makes it trivial to solve them parametrically. Consequently, a great strength of the VARNET library is that it can also be used as a powerful MOR tool. To demonstrate the capabilities of the VARNET library, we focus on the advection-diffusion (AD) equation although the presented approach applies to any PDE that can be solved using the FE method. Discretization-based solutions to the AD-PDE often have stability issues for highly advective problems Hughes and Wells (2005) which are magnified when the model is reduced. Through simulations we demonstrate that unlike traditional MOR methods, our approach does not suffer from such instability issues. (iii)

We also propose a new way to optimally select the points needed to train the NNs that is informed by the feedback obtained from the PDE residual. Our optimal sampling method considerably improves the efficiency of the training and the accuracy of the resulting models. The details of this optimal sampling method can be found in Khodayi-mehr and Zavlanos (2019b).

2. Problem Definition

2.1. Advection-Diffusion PDE

Let $\Omega \subset \mathbb{R}^d$ denote a domain of interest where d is its dimension and let $\mathbf{x} \in \Omega$ denote a location in this domain and $t \in [0, T]$ denote the time variable. Furthermore, consider a velocity vector field $\mathbf{u} : [0, T] \times \Omega \to \mathbb{R}^d$ and its corresponding diffusivity field $\kappa : [0, T] \times \Omega \to \mathbb{R}_+$. Then, the transport of a quantity of interest $c : [0, T] \times \Omega \to \mathbb{R}$, e.g., a chemical concentration, in this domain is described by the advection-diffusion (AD) PDE Hundsdorfer (1996)

$$\dot{c} = -\nabla \cdot (-\kappa \nabla c + \mathbf{u}\, c) + s, \tag{1}$$

where $\dot{c} = \partial c / \partial t$ denotes the time derivative and $s : [0, T] \times \Omega \to \mathbb{R}$ is the source field.

Given an appropriate initial condition (IC) and a set of boundary conditions (BCs), it can be shown that the AD-PDE (1) is well-posed and has a unique solution Reddy (2013). In this paper, we use the following IC and BCs:

$$c(0, \mathbf{x}) = g_0(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega, \tag{2a}$$

$$c(t, \mathbf{x}) = g_i(t, \mathbf{x}) \quad \text{for } \mathbf{x} \in \Gamma_i, \tag{2b}$$

where Γ_i for $i \in \{1, \dots, n_b\}$ denote the boundaries of Ω. Equation (2a) describes the initial state of the field at time $t = 0$ whereas equation (2b) prescribes the field value along the boundary Γ_i as a time-varying function.

We refer to the parameters that appear in the AD-PDE (1), i.e., the velocity, diffusivity, and source fields as well as the IC and BCs, as the input data. Any of these fields can depend on secondary parameters which consequently means that the solution of the AD-PDE will depend on those secondary parameters. For instance, in source identification problems, a parametric solution of the AD-PDE is sought as the properties of the source term $s(t, \mathbf{x})$ vary. This parametric solution is then used to solve a PDE-constrained optimization problem Khodayi-mehr et al. (2019). In general, secondary parameters can also appear in MOR of PDEs. In the following, we denote by $\mathbf{p} \in \mathcal{P} \subset \mathbb{R}^m$ the set of m secondary parameters that the PDE input-data depend on, where $\mathcal{P}$ is the set of feasible values.

2.2. Training Problem

Discretization-based numerical methods, e.g., the FE method, can be used to approximate the solution of the AD-PDE (1). As discussed earlier, these methods although very expressive, are computationally demanding to store and evaluate. Furthermore, it is not straightforward to obtain parametric solutions of PDEs using these methods. Utilizing neural networks (NNs) to solve PDEs, allows us to directly obtain differentiable parametric solutions of PDEs that are computationally more efficient to evaluate and require considerably less memory to store.

Let $\boldsymbol{\theta} \in \mathbb{R}^n$ denote the weights and biases of the NN, a total of n trainable parameters. Then the solution of the AD-PDE can be approximated by the nonlinear output function $f(t, \mathbf{x}; \mathbf{p}, \boldsymbol{\theta})$ of this

NN, where $f : [0,T]\times\Omega \to \mathbb{R}$ maps the spatiotemporal coordinates t and $\mathbf{x}$ to the scalar output field of the PDE, given a value for the secondary parameters $\mathbf{p}$. Note that the input of the NN consists of the coordinates t and $\mathbf{x}$ as well as the parameters $\mathbf{p}$ and its dimension is $1+d+m$. In the following, we drop the arguments $\mathbf{p}$ and $\boldsymbol{\theta}$ whenever they are not explicitly needed.

Problem 1 (Training Problem) *Given the PDE input-data and the NN function $f(\cdot)$ with n trainable parameters, find the optimal value $\boldsymbol{\theta}^* \in \mathbb{R}^n$ such that*

$$\boldsymbol{\theta}^* = \operatorname{argmin} \int_{\mathcal{P}} \int_T \int_\Omega |c(t,\mathbf{x};\mathbf{p},\boldsymbol{\theta}) - f(t,\mathbf{x};\mathbf{p},\boldsymbol{\theta})|^2 \, d\mathbf{x}\, dt\, d\mathbf{p}, \tag{3}$$

where $c(\cdot)$ denotes the true solution of the PDE.

The simplest approach to solve Problem 1 is to use supervised learning to obtain an approximate solution of the PDE using labeled data collected from numerical methods and a loss function defined by objective (3). The NN then acts merely as a memory cell and an interpolator to store the solution more efficiently than storing the values at all grid points. This approach does not alleviate the need for discretization-based methods but instead relies on them. Alternative methods exist that train the NN in an unsupervised way. A typical approach is to consider a loss function defined by the PDE residual obtained when the approximate solution $f(\cdot)$ is substituted in (1). This approach has two problems. First, for the AD-PDE (1) it requires evaluation of second order derivatives of $f(\cdot)$. Estimating a function from its higher order derivatives is inefficient since differentiation only retains slope information and is agnostic to translation. Second, training the differential form of the PDE amounts to learning a complicated field by only considering values at a limited set of points, i.e., a measure-zero set, ignoring correlations in space-time. Next, we propose a different approach that addresses these issues by defining a loss function that relies on the variational form of the PDE.

3. Neural Networks for Solution of PDEs

3.1. Variational Loss Function

The goal of Problem 1 is to learn the parameters $\boldsymbol{\theta}$ of the NN so that for all values of $\mathbf{p}$, the function $f(\cdot)$ approximates the solution of the AD-PDE (1) as well as possible. To capture this, we define a loss function $\ell : \mathbb{R}^n \to \mathbb{R}_+$ based on the variational form of the AD-PDE (1) that reflects how well the function $f(\cdot)$ approximates the solution of (1). Let $v : [0,T]\times\Omega \to \mathbb{R}$ be an arbitrary compactly supported test function. Then, the variational form of the AD-PDE is given as

$$l(c,v) = \int_0^T \int_\Omega \left[\nabla c \cdot (\kappa \nabla v + \mathbf{u}\, v) - c\, \dot{v} - s\, v\right] d\mathbf{x}\, dt = 0. \tag{4}$$

This variational form only requires the first-order spatial derivative and also an integration over a non-zero measure set as opposed to a single point. The test function acts as a weight on the PDE residual and the idea is that if (4) holds for a reasonable number of test functions $v(t,\mathbf{x})$ with their compact supports located at different regions in space-time $[0,T]\times\Omega$, the function $f(\cdot)$ has to satisfy the PDE. A very important feature of the test function $v(t,\mathbf{x})$ is that it is compactly supported. This allows local treatment of the PDE as opposed to considering the whole space-time at once and is the basis of the FE method Hughes (2012).[1]

1. See Khodayi-mehr and Zavlanos (2019b) for the explicit form of the test function $v(t,\mathbf{x})$, the derivation of the variational form, and details on the numerical computation of the variational form (4).

Algorithm 1 VarNet Algorithm

Require: Space-time domain $[0,T]\times\Omega$ and PDE input-data;
Require: Widths of MLP-NN layers and the number of training points $n_v, n_0, n_{b,i},$ and n_p;
Require: Number of training epochs N and weights $\mathbf{w}$ in (5);
1: Generate uniform training points over $[0,T]\times\Omega\times\mathcal{P}$;
2: **for** $e = 1 : N$ **do**
3: **for** $j = 1 : n_p$ **do**
4: Update trainable parameters $\boldsymbol{\theta}_{e,j}$ via an optimizer;
5: **end for**
6: **end for**

Given the variational form (4), we can now define the desired loss function. Consider a set of n_v test functions $v_k(t,\mathbf{x})$ sampling the space-time $[0,T]\times\Omega$, a set of n_0 points $\mathbf{x}_k\in\Omega$ corresponding to the IC, and sets of $n_{b,i}$ points $(t_k,\mathbf{x}_k)\in[0,T]\times\Gamma_i$ for the enforcement of the BCs. Then, for a given set of PDE input-data, specified by $\mathbf{p}$, we define the loss function $\ell:\mathbb{R}^n\times\mathcal{P}\to\mathbb{R}_+$ as

$$\begin{aligned}\ell(\boldsymbol{\theta},\mathbf{p}) = w_1\sum\nolimits_{k=1}^{n_v}|l(f,v_k)|^2 + \frac{w_2}{n_0}\sum\nolimits_{k=1}^{n_0}|f(0,\mathbf{x}_k)-g_0(\mathbf{x}_k)|^2\\ + \frac{w_3}{\bar{n}_b}\sum\nolimits_{i=1}^{n_b}\sum\nolimits_{k=1}^{n_{b,i}}|f(t_k,\mathbf{x}_k)-g_i(t_k,\mathbf{x}_k)|^2,\end{aligned}\tag{5}$$

where $l(f,v_k)$ is defined by (4), $\mathbf{w}\in\mathbb{R}^3_+$ stores the penalty weights corresponding to each term, and $\bar{n}_b=\sum_{i=1}^{n_b}n_{b,i}$ is the total number of training points for the BCs. Normalizing by n_0 and $\bar{n}_b$ makes the weights $\mathbf{w}$ in the loss function (5) independent of the number of training points.

Next, consider a set of n_p points $\mathbf{p}_j\in\mathcal{P}$ sampling the space of the secondary parameters. Then, integrating (5) over the secondary parameters we obtain the total loss function $\ell:\mathbb{R}^n\to\mathbb{R}_+$ as

$$\ell(\boldsymbol{\theta}) = \sum\nolimits_{j=1}^{n_p}\ell(\boldsymbol{\theta},\mathbf{p}_j).\tag{6}$$

This loss function is lower-bounded by zero. Since this bound is attainable for the exact solution of the AD-PDE (1), the value of the loss function is an indicator of how well the NN approximates the solution of the PDE. Training using loss function (6) is an instance of unsupervised learning since the solution is not learned from labeled data. Instead, the training data here are unlabeled samples of space-time and the physics captured by the AD-PDE (1) guides learning of the parameters $\boldsymbol{\theta}$. In that sense, our approach is a model-based method as opposed to a merely statistical method that automatically extracts features and cannot be easily interpreted.

3.2. VarNet Deep Learning Library

The proposed VARNET Algorithm 1 utilizes the loss function (6) to approximate the solution of AD-PDEs. It begins by requiring the properties of the spatial domain Ω and the time horizon T as well as the input data to the AD-PDE. These inputs define an AD-PDE instance that is implemented in the VARNET library using the `ADPDE(domain, diff, vel, source, tInterval, IC, BCs, MORvar)` class where `MORvar` is an instance of `MOR` class for parametric problems. Next, the VARNET Algorithm 1 requires the width of the layers of the multi-layer perceptron (MLP) NN, used to capture the solution of the AD-PDE, as well as the number of training points $n_v, n_0, n_{b,i},$ and n_p.

This information defines the training Problem 1 as an instance of `VarNet(layerWidth, discNum, bDiscNum, tDiscNum)` class. Finally, the number of training epochs as well as the penalty weights in the loss function (5) must be given.

Given the number of training points, in line 1, the algorithm generates a uniform grid over space-time, its boundaries, and possibly the space of secondary parameters $\mathcal{P}$. The training process begins in line 2 where in each epoch the algorithm iterates through all training data. The loop over the samples $\mathbf{p}_j$ of the secondary parameters is performed in line 3. In line 4, the algorithm performs the optimization iteration for $\mathbf{p}_j$ at epoch e, updating the parameters $\boldsymbol{\theta}_{e,j}$ of the NN for all training samples of the space-time and a given set of PDE input-data. The training process is performed in the member function `VarNet.train(epochNum, weight, batchNum)`.

The VARNET library Khodayi-mehr and Zavlanos (2019a) implements Algorithm 1 and contains additional functionalities including data parallelism and extensive tools for report generation during training and post-processing of the trained NNs. In Khodayi-mehr and Zavlanos (2019b) we present more details on the implementation of Algorithm 1 and an extensive set of numerical experiments demonstrating its performance.

4. Numerical Experiments

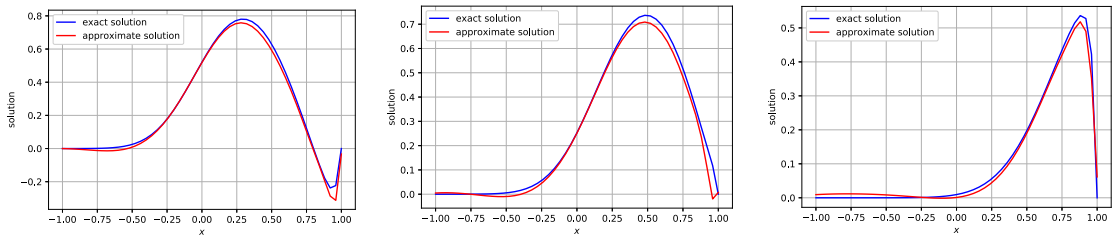
In this section we study the performance of the VARNET Algorithm 1 for the benchmark problem presented in Mojtabi and Deville (2015). This problem is defined for $T = [0, 2]$ and $\Omega = [-1, 1]$ with $s(t, x) = 0$ in (1), $g_0(t, x) = -\sin(\pi x)$ in (2a), and $g_i(t, x) = 0\ \forall i \in \{1, 2\}$ in (2b), respectively. For highly advective cases with large Peclet numbers,[2] a boundary layer is formed whose prediction requires very fine grids when solved by discretization-based numerical methods. In the following results, similar to Mojtabi and Deville (2015), we fix the velocity $u = 1$ and study the performance of our approach for two diffusivity values $\kappa = 0.1/\pi$ and $\kappa = 0.01/\pi$.

We use the ADAM optimization algorithm to train the NNs; see Kingma and Ba (2014). The VARNET Algorithm 1 is run on a single NVIDIA GEFORCE RTX 2080 TI processor for each problem instance. In each case, we use a multi-layer perceptron (MLP) to capture the solution and unless otherwise specified, we use a `sigmoid` activation function for all neurons. For a given uniform grid over the space-time, we compute the approximation error as $\text{err} = \|\mathbf{f} - \mathbf{c}\|/\|\mathbf{c}\|$, where the vectors $\mathbf{f}$ and $\mathbf{c}$ stack the outputs of the NN and exact solution at the grid points, respectively. When the error values are biased against the NN solution, we add an asterisk to the reported value.

4.1. Low Peclet Number

Figure 1 shows the solution, provided by the VARNET Algorithm 1, for $\kappa = 0.1/\pi$ overlaid on the analytical solution. The final error is $\text{err} = 0.04$. We use a single layer MLP with `sigmoid` activation, 20 neurons in the layer, $n_v = 300 \times 20, n_0 = 20,$ and $n_{b,i} = 300$ training points, and set the weights to $\mathbf{w} = [1, 10, 10]$. Note that the number of temporal training points should be sufficiently large to ensure that the dimensionless Courant number, defined as $C = u\Delta t/\Delta x$, is upper-bounded and guarantee the stability of the solution across time Donea and Huerta (2003). An animation of the approximate solution $f(t, x)$ for the case of $\kappa = 0.1/\pi$ is compared to the analytical solution in Khodayi-mehr and Zavlanos (2019c).

2. Peclet number is a measure of the relative dominance of advection over diffusion and is defined as $\text{Pe} = u\hat{l}/\kappa$, where $\hat{l}$ is a characteristic length of Ω, u is a characteristic velocity, and κ is the average diffusivity of the medium.



(*a*) $t = 0.8$ (*b*) $t = 1.0$ (*c*) $t = 1.6$

Figure 1: Snapshots of the NN solution to the AD problem (1) for diffusivity $\kappa = 0.1/\pi$.

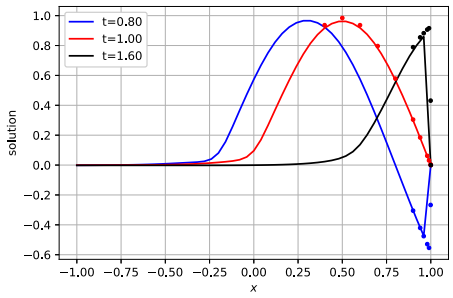


Figure 2: Snapshots of the NN solution for diffusivity $\kappa = 0.01/\pi$ overlaid on exact solution in select points shown by the dots.

Table 1: Error values for diffusivity $\kappa = 0.01/\pi$, calculated for the set of point values reported in Mojtabi and Deville (2015).

No.	n	activation	n_v	err*
1	81	`tanh`	100×10	0.83
2	81	`sigmoid`	100×10	0.49
3	271	`sigmoid`	800×150	0.08
4	911	`sigmoid`	800×150	0.09
5	911	`sigmoid`	1000×200	0.08

As mentioned earlier, one of the important contributions of this paper is the novel loss function based on the variational form of the AD-PDE. To demonstrate the effectiveness of our loss function, we solve the case of $\kappa = 0.1/\pi$ using the differential form of the PDE as well; see also the discussion pursuant to Problem 1. Using the same architecture and settings with $n_v = 1200 \times 80, n_0 = 80$, and $n_{b,i} = 1200$ training points (equivalent to the number of integration points above), the final error is err $= 0.88$. After a set of simulations, the best result corresponding to $n_v = 3000 \times 1000, n_0 = 1000$, and $n_{b,i} = 3000$ and $\mathbf{w} = [1, 10, 5]$ has an error of err $= 0.50$. An animation of the approximate solution $f(t, x)$ for this case is compared to the analytical solution in Khodayi-mehr and Zavlanos (2019e). Observe that there is considerable error in capturing the solution and its BICs when the differential form is used, although the number of training points is much larger.

4.2. High Peclet Number

Next, we consider the case of $\kappa = 0.01/\pi$ which corresponds to an order of magnitude higher Peclet number and is much more challenging to solve. Figure 2 shows the snapshots of the solution from a MLP with two layers with $[10, 20]$ neurons in the layers amounting to $n = 271$ trainable parameters. We train the NN using $n_v = 800 \times 150, n_0 = 150$, and $n_{b,i} = 800$ training points and set the weights to $\mathbf{w} = [1, 10, 10]$. Note that for $\kappa = 0.01/\pi$ the analytical solution is numerically unstable and we only compare to a set of point values reported in Mojtabi and Deville (2015). The final error comparing to these point values is err* $= 0.09$. This error value is conservative since it only considers the boundary layer which is the most challenging to approximate, as opposed to the whole space-time. Discretization-based methods have difficulty capturing the boundary layer at

Table 2: Error values for the parametric solution of the AD problem (1) as a function of diffusivity κ.

No.	n	n_v	κ	err
1	101	300×20	$0.01/\pi \approx 0.0032$	0.33*
2			0.005	0.40*
3			$0.1/\pi \approx 0.03183$	0.14
4	921	800×150	$0.01/\pi$	0.09*
5			0.005	0.15*
6			$0.1/\pi$	0.00

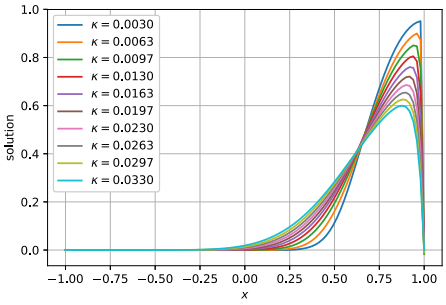


Figure 3: Parametric solution of the AD problem (1) as a function of κ at $t = 1.5$.

$x = 1$ and require an extremely fine grid and consequently a large model, see Mojtabi and Deville (2015), whereas our approach can capture the solution with only $n = 271$ trainable parameters. An animation of the solution $f(t, x)$ for this case is given in Khodayi-mehr and Zavlanos (2019d).

Table 1 shows the performance of the VARNET Algorithm 1, in solving the AD problem (1) with diffusivity $\kappa = 0.01/\pi$, for different activation functions and network capacities. Specifically, we examine three MLP structures with $[20]$, $[10, 20]$, and $[10, 20, 30]$ neurons in the layers. The number of training points n_0 and $n_{b,i}$ for BICs can be deduced from n_v and are not reported. Comparing the first two cases, observe that `tanh` activation is incapable of capturing the solution. From case 3, it can be seen that simultaneous increase of the number of trainable parameters n and the number of training points n_v dramatically decreases the error. Note however that increasing n without increasing n_v destabilizes the training process. Finally considering cases 4 and 5, observe that further increase of the network capacity does not seem to decrease the error. Referring to Figure 2 note that the error values reported in Table 1 effectively capture the error in boundary layer $x = 1$. To capture this layer even better, the number of spatial samples and thus, to maintain the Courant number, the number of temporal samples need to be further increased.

4.3. Model Order Reduction

Finally, we study the performance of the VARNET algorithm 1 for MOR by training NNs that parametrically solve the AD-PDE (1) as a function of diffusivity for $\kappa \in [0.003, 0.033]$, which amounts to a range of more than one order of magnitude. We use $n_p = 5$ geometric samples $\kappa \in \{0.003, 0.0048, 0.0078, 0.0126, 0.0204, 0.033\}$ for training. Table 2 shows the final errors for diffusivity values other than the ones used for training. For the two smaller values of diffusivity, the analytical solution is unstable and only the point values reported in Mojtabi and Deville (2015) are compared which results in biased (larger) error values for these cases. Note that the parametric solutions with only $n = 921$ trainable parameters are as good as the ones reported for individual diffusivity values above, meaning that MOR has no major effect in the solution. This is in contrast to traditional MOR methods that often introduce large errors. The continuous, parametric representation of the solution provides valuable tools for analysis of dynamical systems. For instance, we can study the sensitivity of the solution of the AD problem (1) to the change of diffusivity κ of the medium. Figure 3 shows the solution snapshots at $t = 1.5$ as a function of κ. Note that as κ decreases, the peak value is better preserved and carried downstream with less diffusion and as a result, the boundary layer becomes sharper.

References

William F Ames. *Numerical methods for partial differential equations*. Academic press, 2014.

VI Avrutskiy. Neural Networks catching up with finite differences in solving partial differential equations in higher dimensions. *arXiv preprint arXiv:1712.05067*, 2017.

Jean Donea and Antonio Huerta. *Finite Element methods for flow problems*. John Wiley & Sons, 2003.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional Neural Networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 481–490. ACM, 2016.

Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

Thomas JR Hughes and Garth N Wells. Conservation properties for the Galerkin and stabilised forms of the advection–diffusion and incompressible Navier–Stokes equations. *Computer methods in applied mechanics and engineering*, 194(9-11):1141–1159, 2005.

Willem Hundsdorfer. Numerical solution of advection-diffusion-reaction equations. *Centrum voor Wiskunde en Informatica*, 24:30–41, 1996.

Reza Khodayi-mehr and Michael M. Zavlanos. VarNet deep learning library for the solution of partial differential equations, 2019a. https://github.com/RizaXudayi/VarNet.

Reza Khodayi-mehr and Michael M Zavlanos. VarNet: Variational neural networks for the solution of partial differential equations. 2019b. [Online]. Available: https://arxiv.org/pdf/1912.07443.pdf.

Reza Khodayi-mehr and Michael M. Zavlanos. 1D time-dependent Advection-Diffusion PDE - low Peclet number, 2019c. https://vimeo.com/328756962.

Reza Khodayi-mehr and Michael M. Zavlanos. 1D time-dependent Advection-Diffusion PDE - high Peclet number, 2019d. https://vimeo.com/328759472.

Reza Khodayi-mehr and Michael M. Zavlanos. 1D time-dependent Advection-Diffusion PDE - low Peclet number - residual loss function, 2019e. https://vimeo.com/350138714.

Reza Khodayi-mehr and Michael M Zavlanos. Deep learning for robotic mass transport cloaking. *IEEE Transactions on Robotics*, 2020. doi: 10.1109/TRO.2020.2980176.

Reza Khodayi-mehr, Wilkins Aquino, and Michael M. Zavlanos. Model-based active source identification in complex environments. *IEEE Transactions on Robotics*, 35(3):633–652, 2019. doi: 10.1109/TRO.2019.2894039.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric PDE problems with artificial neural networks. *arXiv preprint arXiv:1707.03351*, 2017.

Diederik P Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-Net: Learning PDEs from data. *arXiv preprint arXiv:1710.09668*, 2017.

Abdelkader Mojtabi and Michel O Deville. One-dimensional linear Advection-Diffusion equation: Analytical and Finite Element solutions. *Computers & Fluids*, 107:189–195, 2015.

M Raissi, P Perdikaris, and GE Karniadakis. Physics-informed Neural Networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Pradeep Ramuhalli, Lalita Udpa, and Satish S Udpa. Finite-element neural networks for solving differential equations. *IEEE Transactions on Neural Networks*, 16(6):1381–1392, 2005.

B Dayanand Reddy. *Introductory functional analysis: with applications to boundary value problems and finite elements*, volume 27. Springer, 2013.

Keith Rudd. *Solving Partial Differential Equations Using Artificial Neural Networks*. PhD thesis, Department of Mechanical Engineering and Material Sciences, Duke University, 2013.

Yazdan Shirvany, Mohsen Hayati, and Rostam Moradian. Multilayer perceptron Neural Networks with novel unsupervised training method for numerical solution of the partial differential equations. *Applied Soft Computing*, 9(1):20–29, 2009.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *arXiv preprint:1708.07469*, 2017.

Shiyin Wei, Xiaowei Jin, and Hui Li. General solutions for nonlinear differential equations: a deep reinforcement learning approach. *arXiv preprint arXiv:1805.07297*, 2018.

Chao Xu, Changlong Wang, Fengzhu Ji, and Xichao Yuan. Finite-Element Neural Network-based solving 3-D differential equations in MFL. *IEEE Transactions on Magnetics*, 48(12):4747–4756, 2012.

Neha Yadav, Anupam Yadav, and Manoj Kumar. *An introduction to neural network methods for differential equations*. Springer, 2015.

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *arXiv preprint arXiv:1901.06314*, 2019.