Physics informed deep-learning approach enhanced by POD for forecasting solutions to time-dependent PDE's

Jennifer Zheng, Tan Bui-Thanh, & Krishnanunni C G

Oden Institute for Computational Engineering and Sciences
Moncrief Summer Internship

August 2021

Abstract

This project focus on forecasting solution of time-dependent PDE's using a neural network based surrogate model. The first stage in this approach is to perform a POD based model reduction to approximate the system solution $u(t, x)$. The solution is represented as a linear combination of orthonormal modes by computing the singular value decomposition of a snapshot matrix. A neural network is then employed to learn the nonlinear map between the POD coefficients $\alpha_r(t)$, an r dimensional output, and the time instants which are 1 dimensional inputs. However, this approach can only predict solutions within the training data set and fails at forecasting the solutions accurately at a future time instant outside the training data set. In order to deal with this issue, we incorporate a physics term in the loss function, which is the discretized equation for the POD coefficients $\alpha(t)$. Thus the method suitably combines data with physics (governing PDE) to arrive at a good surrogate model for the underlying process. Numerical results on a linear advection equation show that by encoding the physics into the training process, one can forecast the solution accurately at a future time instant.

Contents

1 Introduction

Partial differential equations (PDE's) are crucial to modeling physical problems in many fields, and solving PDE's accurately and efficiently has always been a challenge. As data-driven machine learning, especially deep learning, has emerged as a popular technique in many fields. It has also become a powerful technique in solving numerical PDE problems. However, with limited data, purely data-driven DNN model is often not accurate enough leading to erroneous predictions due to the over fitting problem. Therefore, it is imperative to encode the physics into the training process for accurate prediction of the solution. Physics-informed neural networks (PINNs) integrates seamlessly data and mathematical physics models, even in partially understood, uncertain and high-dimensional contexts [9, 4]. PINNs have been found to be well suited for the solution of forward and inverse problems related to several different types of PDEs [9]. Over the past few years, PINNs have been successfully implemented for problems such as simulating vortex-induced vibrations, denoising of 4D- flow magnetic resonance imaging [5], simulation of turbulence [6] etc.

Further, to accelerate the training process, one may incorporate model order reduction [8, 2] into the learning process. In particular, the Proper orthogonal decomposition method has emerged as a popular method for reducing the complexity of computer intensive simulations [1]. In this work, we perform a POD based model reduction to approximate the solution $u(t, x)$ to a time-dependent PDE. A neural network is employed to learn the variation of POD coefficients with respect to time. In order to ensure accurate predictions of the solution outside the domain of the training data, a physics informed term is further incorporated in the loss function. Thus the method suitably combines data with physics (governing PDE) to arrive at a good surrogate model for the underlying process.

2 Problem definition

2.1 1D PDE Problem- The Advection equation

We consider the 1D advection equation as follows:

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = f(x, t) \tag{1}$$

where, $u(x, t)$ is the solution to the PDE at time t and position x.

Now, given the field measured data set $X = \{u(\tilde{x}, t_1),\ u(\tilde{x}, t_2),\ \dots u(\tilde{x}, t_n)\}$, where $t_1,\ t_2,\ \dots t_n$ represents the observation instants and $\tilde{x}$ represents finite observation points in space. Note that matrix X is then known as the snapshot matrix. Our goal is to predict the solution $u(\tilde{x}, t_k)$, where $k > n$ or $k < n$. Note that the case of predicting solution u for $k < n$ is relatively easy since neural networks gives excellent performance with regard to interpolating the solutions inside the data set. However, for the case $k > n$, the neural networks may not be able to give good predictions since this corresponds to extrapolating the solution outside the data set.

2.2 Proper Orthogonal decomposition

To estimate the solution at unknown time steps t_k, we will start with the known solutions to given times, $t_1, t_2, \dots t_n$. We will represent the known data with the snapshot matrix X as discussed above. Note that $X \in \mathbb{R}^{m \times n}$, where m is based on the discretization in the spatial domain. Using the singular value decomposition (SVD), we can decompose the data matrix into two orthogonal matrices U, V and a diagonal matrix Σ.

$$X = U \Sigma V^\top$$

With the size of the given matrix X being $m \times n$, U is an $m \times r$ matrix, Σ is an $r \times r$ matrix, and V is an $r \times n$ matrix. With the orthogonal matrix U, we can rewrite the solutions by separating the time and position variables such that:

$$u(t, x) = \sum_{i=1}^{r} U_i(x) \alpha_i(t) \tag{2}$$

where U_i is the i_{th} column on U. While U_i, the left singular vectors contain the position information at each time, the α_i vectors contain the coefficients to the positions at each time. Therefore, one only needs to estimate the unknown coefficients $\alpha(t) \in \mathbb{R}^r$ in lower dimensional space. In order to predict the solution at unknown time with known position information, our approach to solving the problem is to estimate the α_i vectors at each unknown time. The estimations will be implemented using data-driven machine learning methods to predict α_i at unknown times.

3 Implementation

3.1 Discretization

For demonstration, we create the snapshot matrix X by generating synthetic data. For this, we discretize the advection equation with the upwind scheme, which was first proposed by Courant, Isaacson, and Rees in 1952 [3]. The first-order upwind scheme follows the equation below [7], noted with u_i^n representing the solution at position i and time n:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a \frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 \text{ for } a > 0$$

We perform the upwind scheme at each time t and use it to create the snapshot matrix X.

3.1.1 Matrix representation of the method

For easier computation in later steps, we represent part of the upwind-scheme by a matrix. Instead of the equation from the section above, we use a matrix A as a representation of the position difference among each iteration, specifically, $uA^\top = a * (u_i - u_{i-1})/\Delta x$.

To construct the coefficient matrix A, we first construct a function with the part of the function that we are trying to replace. Each column of A is then constructed with the input of the function being set to the corresponding unit vector.

3.2 Data Generation through Model Order Reduction

To ensure efficiency of the computation, one thing to note is the size of the training and testing data matrices. While we select $\Delta t = 0.0001$ for the time range of 0 to 1 second and $\Delta x = 0.001$ for the discretization of the position variable, we have an input data size of 1001×10001. A data matrix of this dimension would result in significant expense in computation, especially in running the nerual networks, we thus seek to reduce the dimension of the matrix.

We first plot the singular values of the snapshot matrix, which are the diagonal entries of Σ.

We can observe from Figure 1 that there is a significant truncation at the column index of 169. The singular values after column 169 have values close to zero. This suggests that the first 169 singular vectors hold the most important features of the matrix. Thus our first method is to select the first 169 singular vectors from the orthogonal matrices U to construct the training variable α.

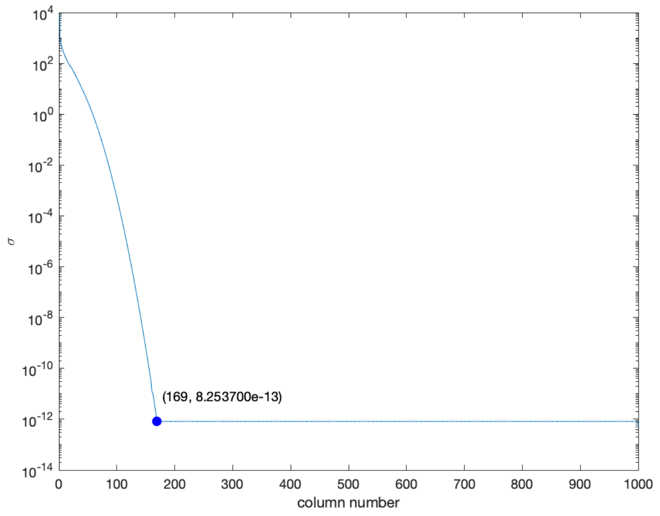


Figure 1: Singular values of the snapshot matrix

The data can be generated by noting that, U in Equation 2 is an orthogonal matrix. Therefore, we have:

$$U_j^T(x)\ u(t,x) = \sum_{i=1}^{r} U_j^T(x) U_i(x) \alpha_i(t), \quad for\ \ any\ \ j$$

Since, $U_j^T(x)U_i(x) = \delta_{ij}$, where δ_{ij} is the Kronecker delta function. We immediately note that in discretized form:

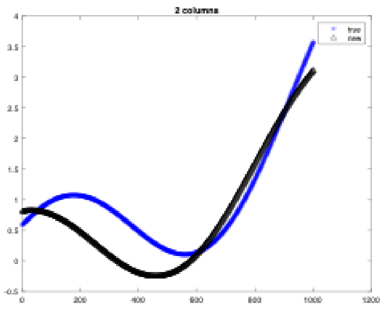
$$U_j^T\ u(t,\tilde{x}) = \alpha_j(t)$$

Thus, generating the data $\alpha_j(t)$. The procedure can be followed for $j = 1,\ 2,\ ...r$, to recover the $\alpha_1(t),\ \alpha_2(t),\ \alpha_3(t) \ldots \alpha_r(t)$. To reevaluate the solution, we explicitly reconstruct α using the forward Euler method:

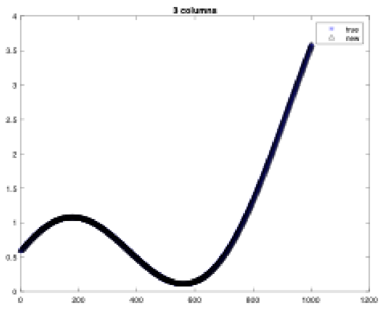
$$U\alpha(t_{n-1}) = U\alpha(t_n) + \Delta t U U^\top A X(t_n) + U U^\top f(t_n)$$

where A was computed in the previous section.

We then plot the reconstruction of the solution, the singular vectors multiplied with the newly constructed α, in comparison with the true solutions. As we evaluate with different number of iterations, we observe that only three singular vectors are necessary to obtain a close approximation of the solution. As shown in Figure 2, the reconstruction using 3 singular vectors greatly overlap with the true solution, and the reconstruction using 2 singular vectors shows significant numerical errors.



(a) 2 singular vectors



(b) 3 singular vectors

Figure 2: Reconstruction of the solution vs. true solution: A look into the effect of choosing different r for model reduction

We thus select the first 3 singular vectors from the orthogonal matrices U to construct the training variable α in many of our numerical experiments later.

3.3 Physics informed neural network

While deep neural network is a commonly-implemented model, it has disadvantages over predicting a solution with output dimension significantly greater than the input dimension. We thus impose a physics term as the model constraint added to the loss function for the deep neural network to improve the prediction. Let us denote $\hat{\alpha} = \{\alpha_1,\ \alpha_2,\ \dots \alpha_r\}$. Then, one condition that has to be satisfied based on the Euler's scheme (implicit) is that the function:

$$f(\theta) = \sum_{t=0}^{t=t_n} \Big(||\hat{\alpha}(t+\Delta t) - \hat{\alpha}(t) + \Delta t U_r^\top A U_r \hat{\alpha}(t+\Delta t)||_2 \Big)^2$$

should approximately equal to 0 where $\hat{\alpha}$ is the solution predicted by the neural network and is a function of the NN parameters θ. We thus choose the residual between the neural network predicted solution and the solution as the physics term.

We create α_1 by multiplying the orthogonal matrix U with all but the last column of the predicted solution, and we create α_0 by multiplying the orthogonal matrix U with all but the first column of the predicted solution.

The physics term is then

$$f(\theta) = \sum_{t=0}^{t=t_n} ||\hat{\alpha}(t+\Delta t) - \hat{\alpha}(t) - \Delta t A \hat{\alpha}(t) - \Delta t \times BC||_2^2$$

where the matrix BC contains the boundary conditions at each time t. The matrix BC is constructed such that each column of t contains the boundary condition at $x = 0$ as the first entry, and the value of $f(x)$ in the rest of the entries. The physics term is added to the loss function, as the mean squared error, with a weight assigned to the physics term.

4 Numerical Experiments and Results

The function we are using as the forcing function of the advection equation in this set of numerical experiments is $f(x,t) = 6ax$ such that the true solution to the PDE is

$$u(x,t) = sin(2\pi(x - at)) + 3x^2.$$

The true solution in later paragraphs will be referring to the solution produced by this function.

4.1 Boundary Conditions and Initial Conditions

As we assume the knowledge of the initial condition and the boundary conditions when we are given a real-world data matrix, the data for our numerical experiments are self-generated. We thus set the boundary conditions and the initial condition to be the true solutions during the discretization step to ensure accuracy.

Figure 3 plots the solution at time $t = 1$ obtained from the upwind-scheme using the true solution as the initial condition and boundary conditions. As the reconstructed numerical solution greatly overlaps with the true solution, we can conclude that this selection of method and boundary conditions accurately discretizes the PDE equation.

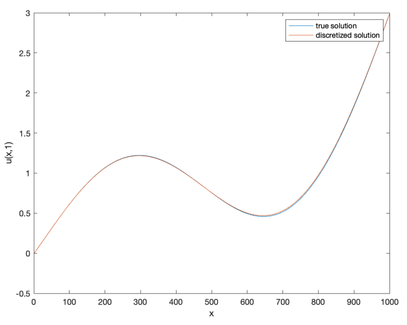


Figure 3: True solution vs. Numerical solution with different boundary conditions

An example of non-ideal boundary condition is the periodic boundary condition. Even though it is commonly used in neural network problems, it shows significant accumulation of errors during the implementation of our problem and would result in Figure 3 (b) below compares the true solution with the numerical solution constructed with periodic boundary conditions. The error is significant in contrast to Figure 3 (a) while the only difference between the two sets of numerical experiments are the boundary conditions.

4.2 Hyperparameters

To tune the PINN to produce the best predictions, there are 6 parameters that we adjust during the numerical experiments.

1. model constraint parameter
2. number of epochs
3. batch size
4. learning rate
5. number of layers
6. number of neurons in each layer

4.3 Prediction within the given time range of training data set

We first test the ability of a basic neural network to predict solutions within the given data range.

In this experiment, we choose $\Delta t = 0.0005$ among the time range of 0 to 1 second. The training data set is created with $t = 0.0005n$ for n being an even integer, and the testing data set is created with n being an odd integer.

As the resulting solutions are presented in a video format, we will choose a time t among the testing data input range to show the comparisons. The sample plot in Figure 4 shows that the predicted solution and the true solution perfectly overlap, suggesting that the DNN is a useful approach.

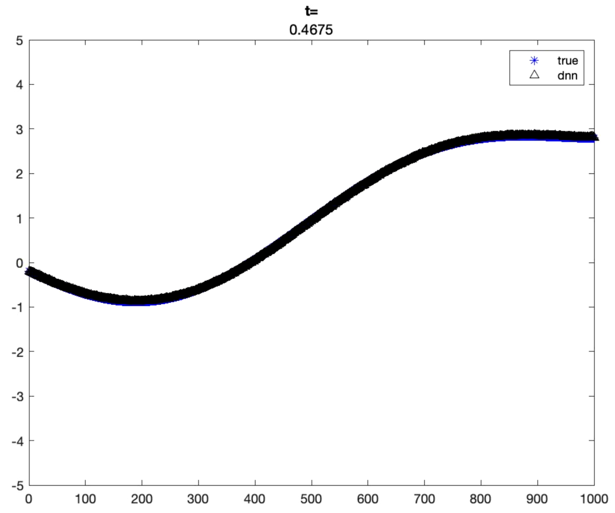


Figure 4: True solution vs. Neural Network predicted solution within given data range

4.4 Predictions outside the time range of the given training data set

4.4.1 Optimal result without physics term

While using the same set of training and testing data set as the previous section, we search for another set of optimal hyperparameters with the model constraint parameter set to 0, which means that this neural network does not contain any physics term.

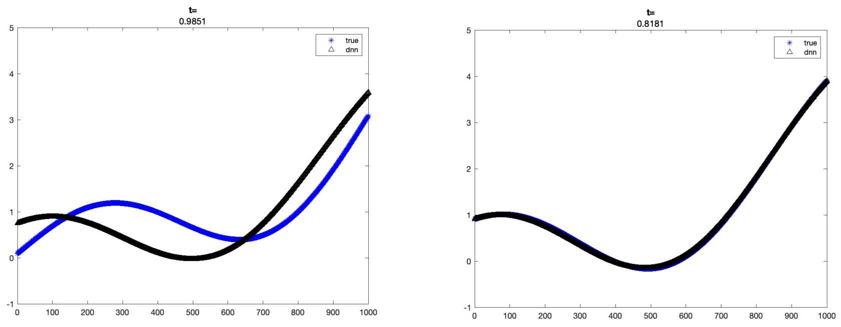
The most optimal parameters in this setting are 4 layers with 7 hidden nodes each, 1000 epochs each with a batch size of 40, and learning rate of 0.001. The resulting prediction is shown in Figure 5 (a). Even with the most optimal set of hyperparameters, the prediction still shows severe deviation from the true solution, suggesting the necessity of adding the physics term.

4.4.2 Optimal PINN result

For predictions outside of the data range, we use $\Delta t = 0.0001$ as the time step, $t = [0, 0.0001, 0.0002, \dots 0.7]$ for creating the training data set, and $t = [0.7001, 0.7002, \dots 1]$ for creating the testing data set.

After testing different hyperparameters, the most optimal model we achieved is a neural network with 800 epochs, each batch size being 50, with 4 layers each with 7 hidden nodes, a learning rate of 0.01, and a model constraint parameter of 0.05.

A solution achieved using the model described above is shown in Figure 5 (b). The model produces a test loss of 0.0282, predicting the solution with great accuracy.



(a) 2 singular vectors

(b) 3 singular vectors

Figure 5: True solution vs. Neural Network in forecasting outside training data set: (a) Without the physics term; (b) With Physics term

4.5 Comparison Across Models

t	Within data range	Outside data range	Model constraint
0.01 s	0.0189	0.0025	0.0043
0.5 s	0.0046	0.0037	0.0028
1.0 s	0.0047	0.0037	0.0028

5 Conclusion

With forecasting the solution of time-dependent PDE being a broad field of study, the implementation of the residual as a physics term used for model constraints in a loss function shows to be an effective method in making an accurate prediction. In future works of this project, we aim to add another variable, velocity, as the input parameter of the neural network. We will also look into comparing the Recurrent Neural Network model with the current approach, and into solving higher dimension PDE problems.

References

[1] Gal Berkooz, Philip Holmes, and John L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual review of fluid mechanics*, 25(1):539–575, 1993.

[2] R. Chris Camphouse, James Myatt, Ryan Schmit, Mark Glauser, Julie Ausseur, Marlyn Andino, and Ryan Wallace. *A Snapshot Decomposition Method for Reduced Order Modeling and Boundary Feedback Control.*

[3] Richard Courant, Eugene Isaacson, and Mina Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications on Pure and Applied Mathematics*, 5(3):243–255, 1952.

[4] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, Mar 2018.

[5] Mojtaba F Fathi, Isaac Perez-Raya, Ahmadreza Baghaie, Philipp Berg, Gabor Janiga, Amirhossein Arzani, and Roshan M D'Souza. Super-resolution and denoising of 4d-flow mri using physics-informed deep neural nets. *Computer Methods and Programs in Biomedicine*, 197:105729, 2020.

[6] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[7] Suhas V. Patankar. *Numerical Heat Transfer and Fluid Flow.* Taylor and Francis, 1980.

[8] Benjamin Peherstorfer and Karen Willcox. Dynamic data-driven reduced-order models. *Computer Methods in Applied Mechanics and Engineering*, 291:21–41, 2015.

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