Title: Understanding machine learning approaches for partial differential equations

Author(s): Gress, Gabriel Jacob

Intended for: Report

Issued: 2020-09-28

Understanding machine learning approaches for partial differential equations

Gabriel Gress, *Student, T-5 LANL, Rice University*

***Abstract*—Recent works in computational physics have been successful in the super-resolution of numerical solutions of partial differential equations (PDEs) via neural networks [1]. In this paper we explore the possibilities and limitations of the data-driven discretization approach implemented by Bar-Sinai et. al., while contrasting it with a simpler yet weaker approach utilizing the nangs Python library. The results demonstrate what neural network parameters optimize accuracy and performance, as well as what conditions on PDEs are necessary to maintain convergence.**

I. Introduction

In the field of computational PDEs, numerical approaches currently face issues of complexity (enormous computer power required for acceptable results) and versatility (often fails except on specific differential equation). In particular, advancing the PDE accurately requires a high number of cells, which does not scale well computationally. As a result, methods that take coarse numerical solutions of differential equations and generate finer solutions have been gaining interest due to works by Bar-Sinai et al. and Mattheakis [1], [2]. There is no doubt to the potential application of such methods, as they allow for large-scale systems to be reasonably tackled with available hardware.

Neural networks have recently proven to have potential in this regard. Neural networks take an input, feed it through layers of neurons with activation functions to yield an output, from which we can evaluate its accuracy through a loss function. Through back-propagation, the gradient of the loss function can then be used to improve the neural network, which can then be used to better model solutions after several iterations. Differential equations translate naturally into machine learning, as the equation itself often serves nicely as a loss function. Furthermore, the machine learning method essentially functions as a universal approximator and is well-suited to modeling non-linear phenomena, and thus can often capture the important dynamics of a differential equation without some caveats to which traditional numerical methods are susceptible. Details on what makes machine learning particularly salient for these problems can be found in the paper by Bar-Sinai et al. [1].

In their paper they introduce a methodology referred to as data-driven discretization. Their method relies on utilizing large quantities of coarse solutions to a differential equation to then build a robust model that can accurately predict the solutions at higher resolutions. This drastically reduces computation time when handling high resolution simulations, but as a result the training process for the model can be expensive. Due to the volatility of solutions by changing the parameters for a differential equation, it requires extensive quantities of data to ensure the model remains robust under any conditions. We investigated approaches to incorporate more knowledge of the underlying PDEs into the model and the generation of data, so that less data is necessary to train the model. By rescaling the data appropriately, we were able to formulate the solutions to be scaled appropriately by parameters such as viscosity in the case of the Burgers' Equation. This constrains the data thereby reducing the training cost.

In contrast to the data-driven discretization approach, the core idea of using the differential equation itself and associated boundary/initial conditions as a loss function does not actually require data beyond the initial conditions to implement. This is the approach the nangs Python library [3] takes, where it approximates the solution entirely by back-propagation from the differential equation via a multi-layer perceptron (MLP) neural network. This has a significant benefit of working without previous numerical computation. It has shown promising results with regards to some simpler differential equations, e.g. the heat equation, as can be seen in the paper corresponding to the library [4]. We explored the capabilities of the library, experimenting with potentially problematic conditions such as discontinuous initial conditions, conservative and non-conservative forms of PDEs, and modifying sampling methods to improve convergence. This method is inherently less versatile than the data-driven approach, but modifying the model with a PDE in mind can help close the gap. For example, the Evans-Krylov theorem bounds the values of convex elliptic differential equations within a unit ball—these bounds can be included into the loss function for these equations, improving convergence [5]. We discuss this approach first, as improvements to the nangs approach can often be applied to the data-driven approach, but it is more difficult to do the reverse.

II. THE MLP METHOD

We used a Multi-Layer Perception (MLP) model to model the differential equation as shown in Fig. 1.

Our first step in exploring the nangs methodology was to reconstruct the generalized approach with the one-dimensional advection equation in mind. During the re-implementation process, we can skip steps that are not relevant to the advection equation, improving performance and allowing us to see the full capabilities of the model (Fig. 2).

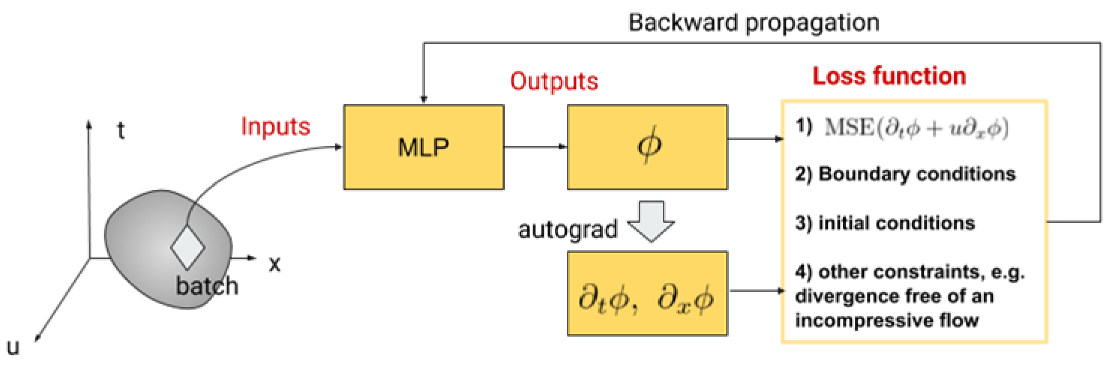
Backward propagation
Outputs
Loss function
Inputs
MLP
ϕ
t
x
u
batch
autograd
$\partial_t\phi,\ \partial_x\phi$
1) $\mathrm{MSE}(\partial_t\phi + u\partial_x\phi)$
2) Boundary conditions
3) initial conditions
4) other constraints, e.g. divergence free of an incompressive flow

Fig. 1. A batch represents a small section of the x-grid and t-grid, from which the model predicts the solution. Then we feed the solution into the differential equation (here the one-dimensional advection equation) and calculate mean squared error (MSE). We also incorporate boundary conditions and initial conditions. The gradient then informs the model what weights need to be changed to minimize the MSE.

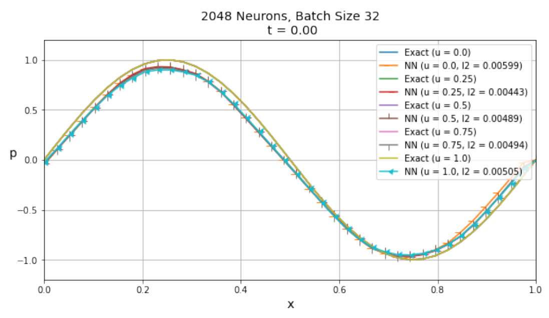

Fig. 2. The neural network solution at $t = 0.00$ of the one-dimension advection equation $\partial_t\phi + u\partial_x\phi = 0$ compared to the exact solution with a sinusoidal initial condition. The purple curve represents the solution with velocity u = 1.0, and the dots represent the neural network's prediction. In this simple case the neural network predicts well, slightly losing accuracy with higher velocities (u).

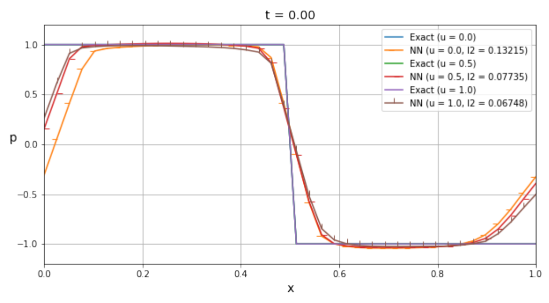

Fig. 3. The same model applied to a periodic unit step function. The model predicts smoother solutions than the actual function at a given time, but error does not propagate as velocity increases, because the L^2 error does not increase for increasing values of u.

We also simultaneously tested jump discontinuities within our initial conditions, to ensure the model can obtain weak solutions (Fig. 3).

Furthermore, we tweaked the parameters of the model to see what neural network structure works best with this approach. Our results were successful, indicating that for a trained model on a given equation, it is robust under modifications. The approach does not rely on a large neural network either — limiting neurons greatly reduces computation time while not affecting overall L^2 error greatly, and increased batch size can decrease both L^2 error and computation time, at the risk of less consistency (Fig. 4).

We observed that the neural network obtains the most error on areas with large gradient. We found that a variant of Chebyshev nodes with higher density on areas of large gradient drastically reduced the error on the variant nodes and maintained equal error on the standard nodes. These results are promising and may prove useful when modeling more complex equations with an adaptive mesh (Fig. 5).

A. *SHORTCOMINGS*

Generalizing the above model proved to be difficult. Replacing the advection equation with the non-conservative form of the equation failed, obtaining mediocre performance compared to the case using the conservative form. No improvements we tried were effective at capturing the proper phenomena (Fig. 6)

III. THE DATA-DRIVEN METHOD

Similar to the nangs method, our first step was to reconstruct the model given in the original paper [1]. This process proved difficult due to changes in code since the publication, and so the results are difficult to compare against the original paper. In the original implementation, the time steps of the data were already normalized to an integer grid — we further normalized the range of values of the solution to prevent the model from fitting to irrelevant factors. Overall error is not reduced, but the training loss decreased faster, and thus requires less training to obtain the same results as the original implementation.

In both cases, the model fails to adhere to boundary and smoothness conditions in early steps. We did not have time to explore an implementation that "blends" the two loss functions, so that the data-driven model also takes into account these conditions (and hopefully converging quicker). However, due to the promising results from the nangs library, we expect that modifications to the model that allow it to take into account properties of the equations will prove effective (Fig. 7)

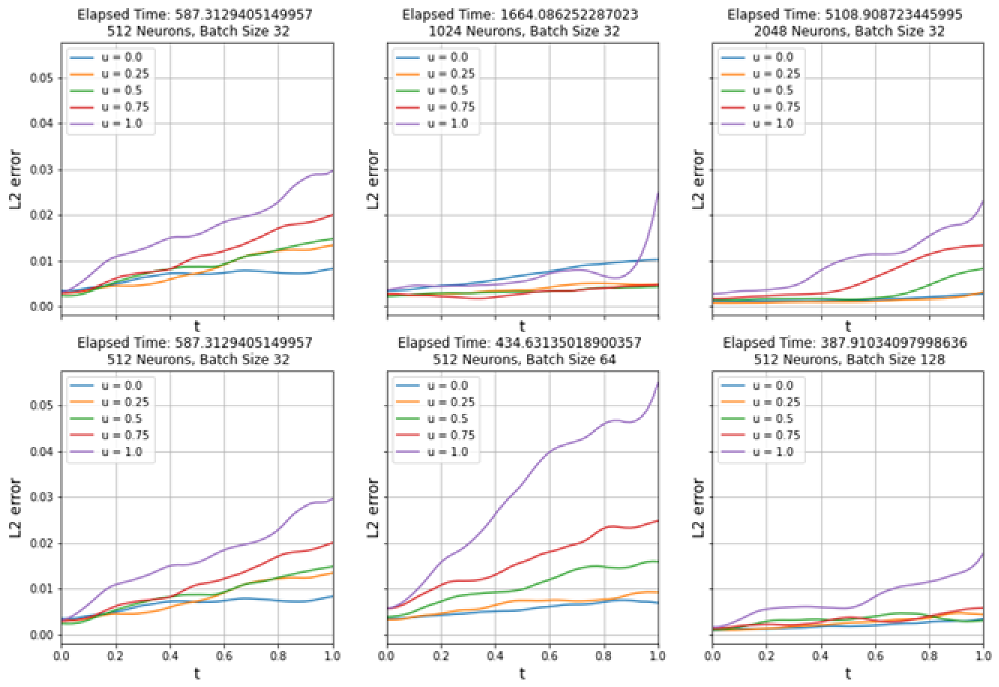

Fig. 4. The difference in L^2 error with increasing neurons (first row from left to right) is minimal, while greatly increasing computation time. Thus, it is preferable to keep the network smaller. However, increasing the batch size (bottom row from left to right) moderately helps computation time and can, in some cases, greatly minimize L^2 error. Further data points are necessary to help identify the relation of batch size to error. The L^2 error also approximately increases linearly with time for small advection velocities.

IV. Future Work

While the approach used in the nangs library is promising for some PDEs, we have yet to see if proper sanitation of data may be effective. Furthermore, the current approach utilizes the bare minimum information of the differential equation — using a more informed model may be able to capture dispersion and shock effects properly. The model would also need to be tested on broader classes of differential equations. Further exploration may find other classes of equations that are solved better with this method.

The data-driven approach already has been tested on various equations and has been shown to be robust, but there is lots of potential for increases in speed and accuracy. As mentioned prior, using boundary conditions and enforcing smoothness are simple steps that may be effective in improving the model.

Finally, both methods have not been tested with adversarial attacks (e.g. inherent noise in the modeling in this context). The current structure of both approaches is likely vulnerable to these attacks, and as a result are unlikely to model stochastic differential equations well. An approach that is robust under adversarial attacks would prove incredibly useful — and in fact may show effective even in the noiseless case.

Acknowledgment

This work was funded through the Department of Energy's Student Undergraduate Laboratory Internship program hosted by Los Alamos National Laboratory with the guidance and advice from Chengkun Huang, Feiyu Li, Platon Karpov, and Gyrya Vitaliy. This project would not have been possible without their support. This work is also partially supported by the LANL LDRD project 20190131ER.

References

[1] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 31, pp. 15 344–15 349, 2019. [Online]. Available: https://www.pnas.org/content/116/31/15344

[2] M. Mattheakis, D. Sondak, A. S. Dogra, and P. Protopapas, "Hamiltonian Neural Networks for solving differential equations," *arXiv e-prints*, p. arXiv:2001.11107, Jan. 2020.

[3] J. B. Pedro, "Solving pdes with nns," https://github.com/juansensio/nangs, 2019.

[4] J. B. Pedro, J. Maroñas, and R. Paredes, "Solving Partial Differential Equations with Neural Networks," *arXiv e-prints*, p. arXiv:1912.04737, Dec. 2019.

[5] L. Caffarelli and L. Silvestre, "On the Evans-Krylov theorem," *Proceedings of the American Mathematical Society*, vol. 138, no. 1, pp. 263–265, jan 2010. [Online]. Available: http://www.ams.org/jourcgi/jour-getitem?pii=S0002-9939-09-10077-1

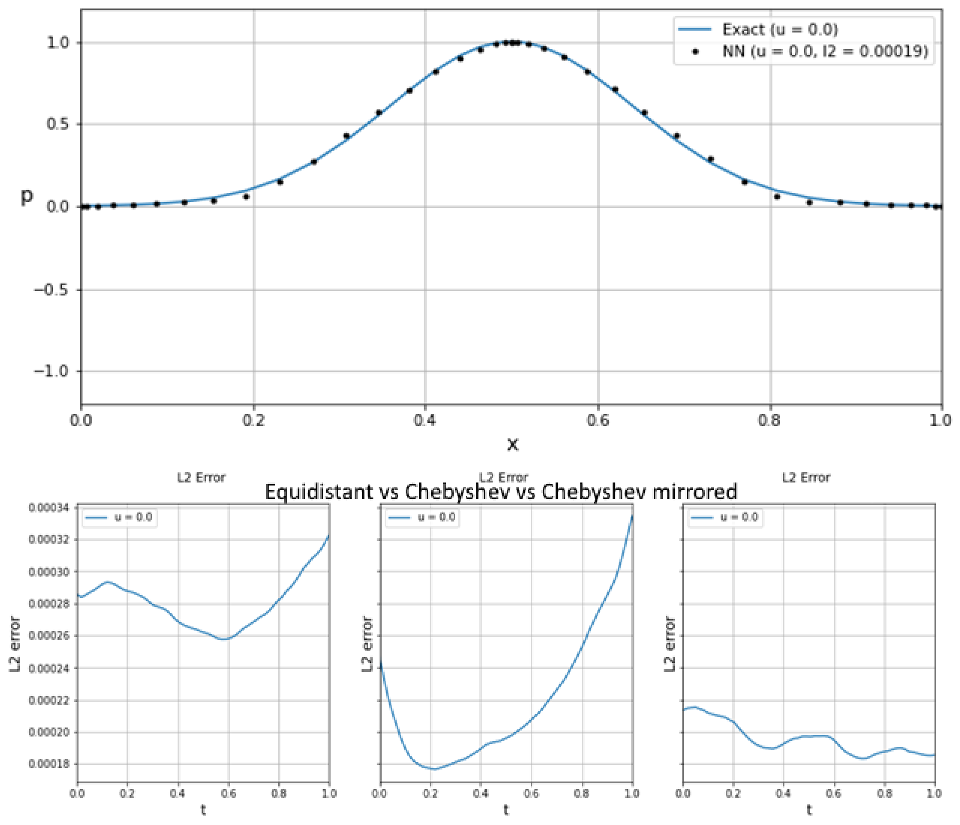

Fig. 5. Despite that the MLP model is incognizant of the choice of the mesh points, using variant of Chebyshev nodes (black dots) greatly improve error on the numerical solution (bottom panels) as well as distributing it better throughout the grid for the stationary case $u = 0$. In the bottom panels, the first graph represents the standard equidistant nodes. The second graph represents the standard Chebyshev nodes, with no worthwhile improvement over the standard. The third graph contains the variant Chebyshev nodes, which both reduce L^2 error significantly, as well as reduce the variation for a given value of t.

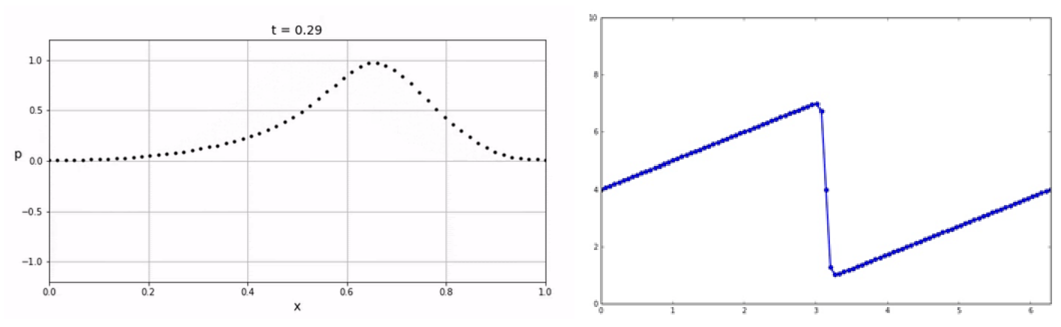

Fig. 6. The first figure is the neural network prediction for the solution of the Inviscid Burgers' Equation. It fails to capture the expected shock behavior, as depicted in the second diagram. This indicates the neural network is failing to capture the desired behavior.

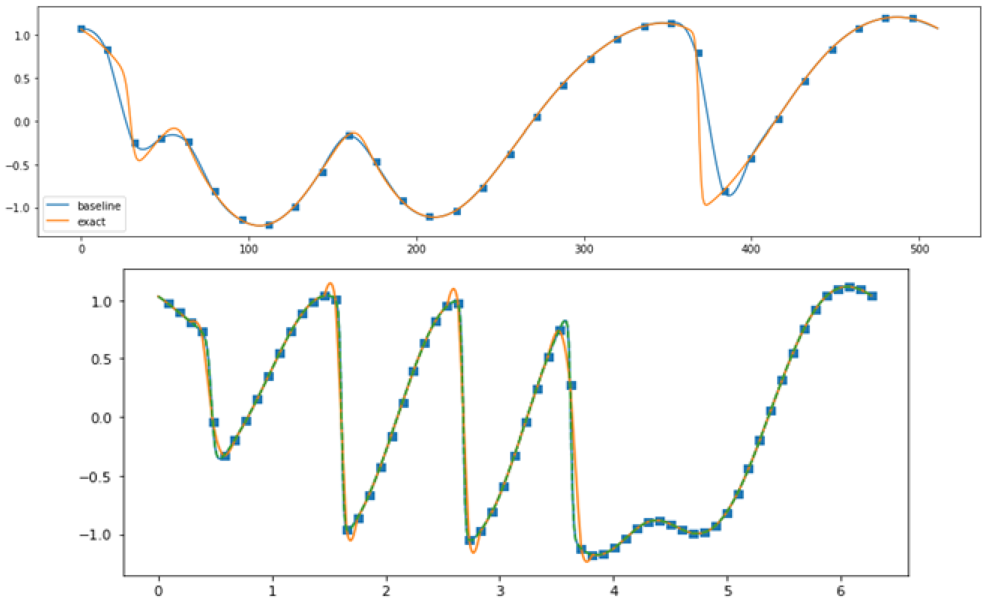

Fig. 7. The graphs depict a solution of the Inviscid Burgers' Equation through the data-driven discretization approach. The baseline is the interpolation of the neural network after training. Initially it fails to capture the sharper aspects of the curve, but the second diagram shows that it converges well to the complex behavior of the differential equation. The graphs were obtained using the normalized data and show a marked improvement over the original results.