The Journal of Open Source Software

nnde: A Python package for solving differential equations using neural networks

Eric Winter[1] **and R.S. Weigel**[1]

1 Department of Physics and Astronomy, George Mason University

DOI: 10.21105/joss.0XXXX

Software

- Review
- Repository
- Archive

Editor: Editor Name

Submitted: 01 January XXXX
Published: 01 January XXXX

Summary

Neural networks have been shown to have the ability to solve differential equations (Chakraverty & Mall, 2017; Yadav et al., 2015). `nnde` is a pure-Python package for the solution of ordinary and partial differential equations of up to second order. We present results of sample runs showing the effectiveness of the software in solving the two-dimensional diffusion problem.

Statement of need

The `nnde` package provides a pure-Python implementation of one of the earliest approaches to using neural networks to solve differential equations - the trial function method (Lagaris et al., 1998). It was initially developed primarily as a vehicle for understanding the internal workings of feedforward neural networks, without the restrictions imposed by an existing neural network framework. It has since been enhanced to provide the capability to solve differential equations of scientific interest, such as the diffusion equation described here. The ultimate goal of the package is to provide the capability to solve systems of coupled partial differential equations, such as the equations of magnetohydrodynamics. Current high-performance neural network packages, such as TensorFlow or PyTorch, can improve the speed of the matrix computations required for the neural network solutions. But these packages cannot perform the analytical derivatives required by the algorithms implemented in this code.

The most commonly used methods for solving differential equations are the Finite Element Method (FEM) and Finite Difference Method (FDM). However, these methods can be difficult to parallelize and may have large storage requirements for model outputs. The neural network method is straightforward to parallelize due to the independent characteristics of the computational nodes in each network layer. Additionally, the trained network solution is more compact than an FDM or FEM solution because storage of only the network weights and biases are required. The neural network solution is mesh-free and does not require interpolation to retrieve the solution at a non-grid point, as is the case with FDM or FEM. Once the network is trained, computing a solution at any spatial or temporal scale requires only a series of matrix multiplications, one per network layer. The trained solution is a sum of arbitrary differentiable basis functions, and therefore the trained solution is also differentiable, which is particularly useful when computing derived quantities such as gradients and fluxes.

Description

`nnde` implements a version of the trial function algorithm described by Lagaris et al. (1998). This software also incorporates a modification of the trial function algorithm to automatically

incorporate arbitrary Dirichlet boundary conditions of the problem directly into the neural network solution.

This software was developed as part of a project to demonstrate that the coupled partial differential equations of magnetohydrodynamics (MHD) (Chen, 1984) can be solved using neural networks. The result is a package can be used for any ordinary or partial differential equations of up to second order. `nnde` was initially developed entirely in Python to understand the workings of the neural network algorithm.

Consider the diffusion equation in two dimensions as an example of the type of problem that can be solved with `nnde`.

$$\frac{\partial \psi}{\partial t} - D\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}\right) = 0$$

With all boundaries fixed at 0 and with an initial condition of

$$\psi(x, y, 0) = \sin(\pi x)\sin(\pi y)$$

the analytical solution is

$$\psi_a(x, y, t) = e^{-2\pi^2 Dt}\sin(\pi x)\sin(\pi y)$$

The `nnde` package was used to create a neural network with a single hidden layer and 10 hidden nodes and trained to solve this problem. The error in the trained solution for the case of $D = 0.1$ is shown as a function of time in Figure 1.

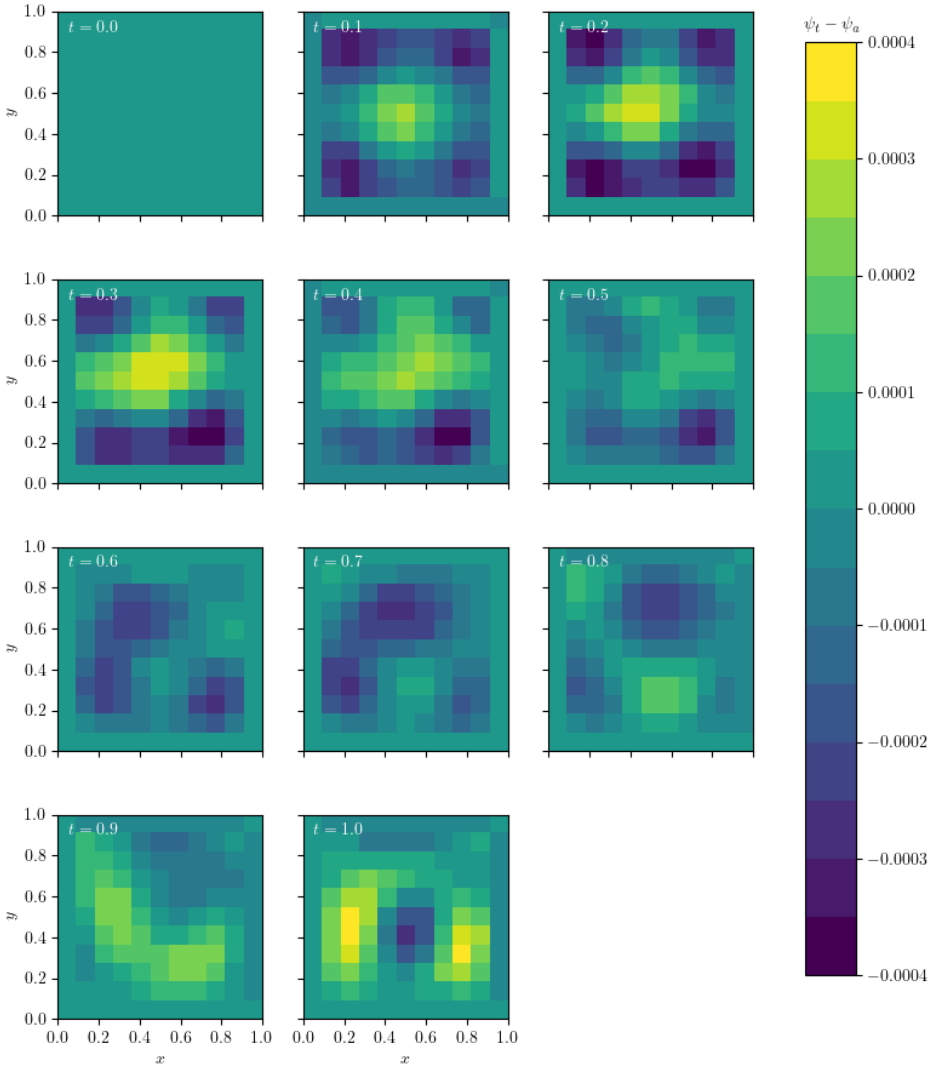

Figure 1: Difference between the trained neural network solution $\psi_t(x, y, t)$ and the analytical solution $\psi_a(x, y, t)$ of the diffusion problem in 2 spatial dimensions using `nnde` with 10 nodes.

Software repository

The `nnde` software is available at https://github.com/elwinter/nnde.

A collection of example python scripts using `nnde` is available at https://github.com/elwinter/nnde_demos.

A collection of example Jupyter notebooks using `nnde` is available at https://github.com/elwinter/nnde_notebooks.

References

Chakraverty, S., & Mall, S. (2017). *Artificial neural networks for engineers and scientists: Solving ordinary differential equations*. CRC Press. ISBN: 1498781381

Chen, F. F. (1984). *Introduction to plasma physics and controlled fusion 2e*. Plenum. ISBN: 0-306-41332-9

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, *9*(5), 987–1000. https://doi.org/10.1109/72.712178

Yadav, N., Yadav, A., & Kumar, M. (2015). *An introduction to neural network methods for differential equations*. Springer Netherlands. https://doi.org/10.1007/978-94-017-9816-7