



Ben Stevens · Tim Colonius

Enhancement of shock-capturing methods via machine learning

Received: 31 October 2019 / Accepted: 21 April 2020 / Published online: 23 May 2020
© Springer-Verlag GmbH Germany, part of Springer Nature 2020

Abstract In recent years, machine learning has been used to create data-driven solutions to problems for which an algorithmic solution is intractable, as well as fine-tuning existing algorithms. This research applies machine learning to the development of an improved finite-volume method for simulating PDEs with discontinuous solutions. Shock-capturing methods make use of nonlinear switching functions that are not guaranteed to be optimal. Because data can be used to learn nonlinear relationships, we train a neural network to improve the results of a fifth-order WENO method. We post-process the outputs of the neural network to guarantee that the method is consistent. The training data consist of the exact mapping between cell averages and interpolated values for a set of integrable functions that represent waveforms we would expect to see while simulating a PDE. We demonstrate our method on linear advection of a discontinuous function, the inviscid Burgers' equation, and the 1-D Euler equations. For the latter, we examine the Shu–Osher model problem for turbulence–shock wave interactions. We find that our method outperforms WENO in simulations where the numerical solution becomes overly diffused due to numerical viscosity.

Keywords Shock capturing · Machine learning · Fluid mechanics

1 Introduction

For some initial-boundary value problems (IBVP) in fluid mechanics, the solution of the partial differential equations (PDEs) includes discontinuous initial data or a discontinuity that forms in finite time, i.e., shock waves. Numerical methods for solving these PDE must be specially tailored to properly resolve these discontinuities [19].

These shock-capturing methods are designed with the goal of sharply resolving a shock without inducing spurious oscillations, while also giving accurate solutions in smooth regions of the flow. One major breakthrough in this effort was the development of high-resolution methods [10], as these methods were capable of achieving second-order accuracy without introducing spurious oscillations around shocks. These methods gave rise to a class of high-resolution methods called essentially non-oscillatory (ENO) schemes [11] that measure the smoothness of the solution on several stencils, and then compute the flux based on the smoothest stencil to avoid interpolating through the discontinuity. These schemes are nonlinear (even when the PDEs are linear) since the interpolation coefficients depend on the solution. These ideas were then modified to create

Communicated by Steven L. Brunton.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1745301.

B. Stevens (✉) · T. Colonius
Department of Mechanical and Civil Engineering, California Institute of Technology, Pasadena, CA 91125, USA
E-mail: bstevens@caltech.edu

WENO-JS (weighted ENO-Jiang Shu) methods [13], which again compute the smoothness on several stencils. However, instead of taking only the smoothest stencil, these methods take a weighted average of the fluxes predicted on each stencil to emphasize the smoother ones. When each stencil is equally smooth, the weights are designed to cause the method to converge to the constant coefficient scheme that maximizes the order of accuracy over the union of the sub-stencils, which gives these methods a high order of accuracy for smooth solutions.

Many efforts have been built on the original WENO-JS schemes by modifying the smoothness indicators [9, 14, 24], modifying the nonlinear weights [2, 3, 25], and using WENO-JS as part of a hybrid scheme [20, 22]. While some of these references build off of each other rather than starting from WENO-JS, we will base our method on WENO-JS because our strategy for developing the method does not resemble other methods. However, our methodology could easily adopt various improvements that have been made to WENO-JS.

One commonality that has persisted since the original ENO scheme is a reliance on human intuition in shock-capturing method design, particularly in the nonlinear aspects of the schemes, i.e., smoothness indicators and weighting functions. While they have been well studied, there is no reason to believe that they are optimal. Efforts have been made to develop optimal spatial discretization methods by minimizing wave propagation errors [15, 18, 21, 28] and minimizing error over certain frequency ranges [31], and some of these techniques have even been combined with shock-capturing schemes [7, 29]. However, designing the optimization problem still requires human intuition with regards to balancing competing goals, rather than attempting to learn an optimal scheme from data in an unbiased way.

Over the past decades, machine learning has become ubiquitous in data analysis and is increasingly seen as having potential to improve (or reformulate) numerical methods for PDEs. Dissanayake et al. [6] exploit the fact that neural networks are universal approximators to transform the problem of solving a PDE into an unconstrained minimization problem. Lagaris et al. [17] parameterized the solution to a PDE as a neural network and optimized the weights to minimize the residual of the solution. Yu et al. [30] trained a neural network to classify the local smoothness and apply artificial viscosity based on this classification. Bar-Sinai et al. [1] used simulation data to embed coarse-graining models into finite difference schemes involving neural networks, allowing them to achieve low error on relatively coarse grids. Pfau et al. [23] parameterized the eigenfunctions of eigenvalue problems as a neural network and cast the training as a bilevel optimization problem to reduce bias, resulting in significantly decreased memory requirements. Hsieh et al. [12] attempted to learn domain-specific fast PDE solvers by learning how to iteratively improve the solution using a deep neural network, resulting in a 2-speedup compared to state-of-the-art solvers.

In the current work, we attempt to train a neural network to improve WENO5-JS. Our goal is to get closer to the optimal nonlinear finite-volume coefficients while introducing a minimal amount of bias. Unlike other references, we do not directly change the smoothness indicators or nonlinear weights of the method. Instead, we use a neural network to perturb the finite-volume coefficients determined using the original smoothness indicators and nonlinear weights of WENO5-JS. We attempt to learn an optimal function for this perturbation using data generated from waveforms that are representative of solutions of PDEs. These modifications result in a finite-volume scheme that diffuses fine-scale flow features and discontinuities less severely than WENO5-JS. We start in the next section by giving a more detailed description of the proposed algorithm.

2 Numerical methods

2.1 Description of WENO-NN

Although we focus on WENO5-JS in this paper, our approach could generally be used to enhance any shock-capturing method (or perhaps any numerical method). The proposed algorithm involves preprocessing the flow variables on a stencil using a conventional shock-capturing method and feeding those results into a neural network. The neural network then perturbs the results of the shock capturing method. Post-processing is then applied to the output of the neural network to guarantee consistency [1] (or, more generally, could be used to enforce other desirable properties). Hence, the augmented numerical scheme takes on many properties of the original. For example, applying the method to WENO5-JS results in an upwind-biased finite volume method with coefficients that depend on the local solution. The steps of the algorithm for enhancing WENO5-JS are shown in Algorithm 1.

We use WENO5-JS to preprocess the input data, so that the input to the neural network is the set of finite-volume coefficients found by WENO5-JS. We found that including this preprocessing step significantly

Algorithm 1 WENO-NN Algorithm

-
- 1: **procedure** WENONN
 - 2: Begin with cell averages $\bar{u}_{j-2:j+2}$
 - 3: Scale the cell averages
 - 4: Compute coefficients $\tilde{c}_{j-2:j+2}$ with WENO5-JS
 - 5: Compute change in coefficients $\Delta\tilde{c}_{j-2:j+2}$ with neural network
 - 6: Compute new coefficients $\hat{c}_{j-2:j+2} = \tilde{c}_{j-2:j+2} - \Delta\tilde{c}_{j-2:j+2}$
 - 7: Compute final coefficients $c_{j-2:j+2}$ by transforming $\hat{c}_{j-2:j+2}$
 - 8: Compute cell edge value $u_{j+1/2} = c_{j-2:j+2} \cdot \bar{u}_{j-2:j+2}$
-

improved performance. Once the nonlinear weights w_i are determined according to the WENO5-JS algorithm, the coefficients for each cell average are computed as

$$\begin{aligned}
 \tilde{c}_{-2} &= \frac{1}{3}w_1, \\
 \tilde{c}_{-1} &= -\frac{7}{6}w_1 - \frac{1}{6}w_2, \\
 \tilde{c}_0 &= \frac{11}{6}w_1 + \frac{5}{6}w_2 + \frac{1}{3}w_3, \\
 \tilde{c}_1 &= \frac{1}{3}w_2 + \frac{5}{6}w_3, \\
 \tilde{c}_2 &= -\frac{1}{6}w_3.
 \end{aligned} \tag{1}$$

These five coefficients are the inputs to the neural network, which outputs a change in each coefficient, $\Delta\tilde{c}_i$. Our neural network uses three hidden layers, each with three neurons. We deliberately make the network as small as possible to reduce the computational cost of evaluating it. We are able to use such a small network because assuming that the WENO5-JS coefficients are a useful model input is a strong prior, so WENO5-JS performs a significant amount of the required processing. L_2 regularization is applied to the output of the neural network to penalize deviations from WENO5-JS, which encourages the network to only change the answer supplied by WENO5-JS when an improved result is expected. The new coefficients are computed by subtracting the change in coefficients from the old coefficients.

Additionally, the size of the input space is reduced by scaling cell averages within the stencil as

$$\bar{\mathbf{u}}_s = \frac{\bar{\mathbf{u}} - \min \bar{\mathbf{u}}}{\max \bar{\mathbf{u}} - \min \bar{\mathbf{u}}}. \tag{2}$$

If all the cell averages have the same value, the scaling equation fails, so the value at the cell edge is simply set to the cell average value.

To guarantee that WENO-NN is consistent, we apply an affine transformation to these coefficients that guarantees that they sum to one [1]. We derive this transformation by solving the optimization problem:

$$\begin{aligned}
 \min_{\mathbf{c} \in \mathbb{R}^5} & \sum_{n=-2}^2 (c_n - \hat{c}_n)^2 \\
 \text{s.t.} & \sum_{n=-2}^2 (c_n) = 1,
 \end{aligned} \tag{3}$$

which can be reformulated with the substitution $\Delta\mathbf{c} = \mathbf{c} - \hat{\mathbf{c}}$ to pose the problem as finding the minimum norm solution to an under-constrained linear system

$$\begin{aligned}
 \min_{\Delta\mathbf{c} \in \mathbb{R}^5} & \sum_{n=-2}^2 (\Delta c_n)^2 \\
 \text{s.t.} & \sum_{n=-2}^2 (\hat{c}_n + \Delta c_n) = 1,
 \end{aligned} \tag{4}$$

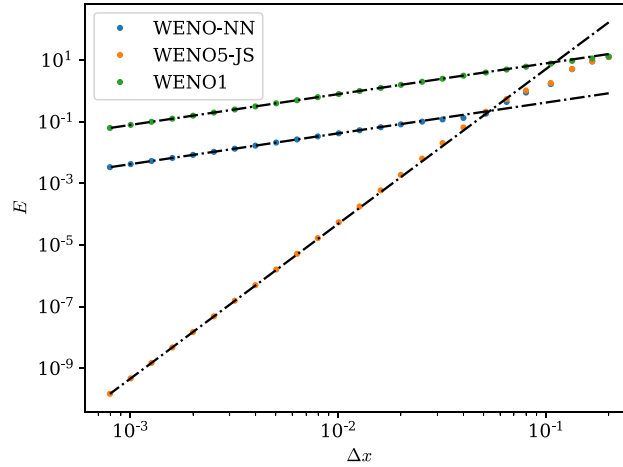


Fig. 1 Convergence of WENO-NN, WENO5-JS, and WENO1 for smooth solutions

which has the analytical solution

$$\Delta c_i = \frac{1 - \sum_{n=-2}^2 \hat{c}_n}{5}. \tag{5}$$

One can use the same approach to enforce arbitrarily high orders of accuracy since the optimization problem has an analytical solution for any constraint matrix of sufficiently high rank

$$\begin{aligned} \min_{\Delta \mathbf{c} \in \mathbb{R}^5} & \sum_{n=-2}^2 (\Delta c_n)^2 \\ \text{s.t. } & A(\hat{\mathbf{c}} + \Delta \mathbf{c}) = \mathbf{b}. \end{aligned} \tag{6}$$

This optimization problem has analytical solution $\Delta \mathbf{c} = A^T (AA^T)^{-1} (\mathbf{b} - A\hat{\mathbf{c}})$ when AA^T is invertible.

We verify that our constraint is satisfied by looking at the convergence rate of WENO-NN for a smooth solution. For this test case, we will simply use WENO-NN, WENO5-JS, and WENO1 to take the derivative of $u(x) = \sin(4\pi x) + \cos(4\pi x)$, and compare the results to the analytical solution $\frac{\partial u}{\partial x}^* = -4\pi \sin(4\pi x) + 4\pi \cos(4\pi x)$ using the error metric

$$E = \sqrt{\frac{\| \frac{\partial u}{\partial x} - \frac{\partial u}{\partial x}^* \|_2}{N}}. \tag{7}$$

In Fig. 1, we can see that WENO-NN achieves the first-order accuracy, which confirms that the constraint is satisfied. We also see that, as expected, WENO5-JS converges at fifth order and WENO1 converges at first order as $\Delta x \rightarrow 0$. However, when discontinuities are present it is not possible to achieve better than first-order accuracy with any finite volume method [19]. Despite this fact, it is advantageous to use WENO5-JS over WENO3-JS in such situations, as WENO5-JS still tends to give lower error in discontinuous problems [26], which is why we chose to use WENO5-JS for processing the cell average values despite the fact that WENO-NN ends up being first-order accurate. Similarly, we see that for some discontinuous problems, WENO-NN gives lower error than WENO5-JS. If a higher order of accuracy is desired in smooth regions of the flow, one could develop a hybrid method using WENO-NN and any high-order method.

2.2 Other numerical methods used

For all simulations shown, we use a third-order TVD Runge–Kutta scheme [8] as our time-stepping method

$$u^{(1)} = u^{(n)} + \Delta t L(u^{(n)}),$$

$$\begin{aligned}
u^{(2)} &= \frac{3}{4}u^{(n)} + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L(u^{(1)}), \\
u^{(n+1)} &= \frac{1}{3}u^{(n)} + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L(u^{(2)}).
\end{aligned} \tag{8}$$

For flux-splitting, we use a Lax–Friedrichs flux splitting procedure: [27]

$$\begin{aligned}
f^\pm(u) &= \frac{1}{2}(f(u) \pm \alpha u), \\
\alpha &= \max_u |f'(u)|.
\end{aligned} \tag{9}$$

In this expression, $f(u)$ is defined as the flux of a 1-D hyperbolic conservation law $\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0$. When solving the 1-D Euler equations, we apply the flux splitting to the characteristic decomposition of the system. For our numerical Riemann solver, we use the Lax–Friedrichs method [5].

3 Machine learning methodology

We construct our training data directly from known functions that we expect to represent the waveforms that WENO-NN will encounter in practice. Note that the same dataset is used for every PDE, as we train only one network and use it for every WENO-NN result shown in this paper. However, one could develop a problem-specific dataset if desired. For each data point, we start with some function $u(x)$ and a discretized domain of n cells. The cell average is evaluated on each cell as

$$\bar{u}(x_i) = \frac{1}{\Delta x} \int_{x_i - \frac{\Delta x}{2}}^{x_i + \frac{\Delta x}{2}} u(x) dx, \tag{10}$$

and because we chose the form of $u(x)$ we can evaluate the cell average analytically. We also evaluate the function value on the cell boundary as $u(x_i + \Delta x/2)$ analytically. We then move along the domain and form the dataset based on the stencil size. So for WENO-NN, one data point involves 5 cell averages as the input with the function value on the cell boundary as the output. The functions we use when creating the dataset are step functions, sawtooth waves, hyperbolic tangent functions, sinusoids, polynomials, and sums of the above. Sinusoids and polynomials broadly cover most smooth solutions that would be encountered in practice, and we specifically chose hyperbolic tangent functions because they mimic smeared out discontinuities. Similarly, step functions and sawtooth waves mimic contact discontinuities and shocks. We included sums of these functions to mimic complex flow fields that may contain shocks within a turbulent flow. We found that including more data made the resulting scheme less diffusive and less oscillatory.

When adding a new entry to the dataset, we first check to see if it is close to other points already present in the L_2 sense. Sufficiently close points are not added to the dataset to prevent redundant data that will slow down the training process. The resulting dataset has 75,241 entries.

When training the network, we use the Adam optimizer [16], split the data into batches of 80 points to estimate the gradient, and optimize for 10 epochs using the Keras package in Python [4]. We trained the network from many different randomly chosen initial guesses of the parameters and chose the best one based on performance in simulating the linear advection of a step function. We apply L_2 regularization with a constant of $\lambda = 0.1$ to the neural network output and find that when splitting the data into a training set of 80% of the data and a validation set of the other 20% of the data our in-sample error is 0.569 and the out-of-sample error is 0.571, averaged from 100 trials of training on the dataset, so overfitting within the generated dataset is not a concern. This difference is so small because the model we are training is of relatively low complexity and is essentially underfitting the generated dataset. We use mean squared loss as our objective function to minimize.

Despite the fact that we do not see overfitting within the generated dataset, we still observe overfitting when we apply the method to an actual simulation. Figure 2 shows the average training error, average validation error, and average error when using the method to simulate a PDE for different regularization values λ of the neural network output. The training and validation errors are computed using the mean square error:

$$E_{\text{data}} = \frac{\sum_{i=1}^N (y_i - y_i^*)^2}{N}, \tag{11}$$

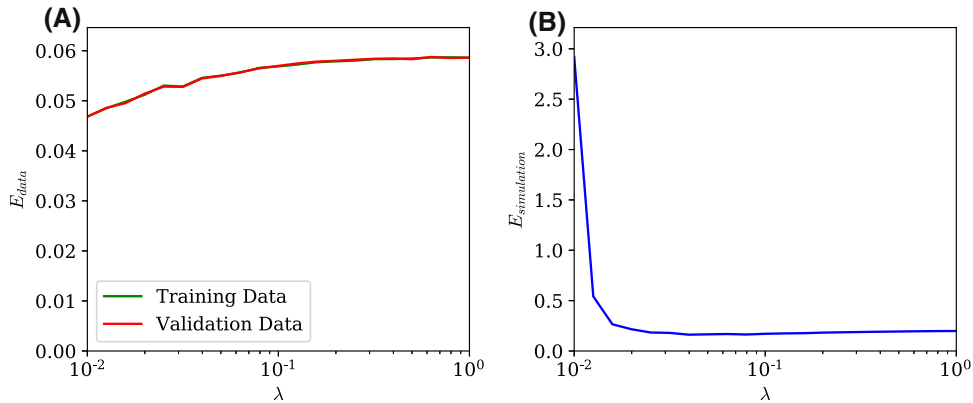


Fig. 2 Comparing error trends between **a** exact generated data and **b** simulation results

while the simulation error is computed by using the learned numerical method to linearly advect a step function and computing the L_2 error at the end of the simulation:

$$E_{simulation} = \sqrt{\int_0^L (\bar{u}(x, T) - \bar{u}^*(x, T))^2 dx}. \quad (12)$$

One can see that adding regularization causes error to increase in both the training and validation datasets, but decreases the error in the simulation results. Hence, we can see that we are overfitting to the training data, but because the validation data do not show this, we can conclude that the dataset does not exactly match the distribution we are trying to approximate.

The following paragraph describes our model development process and can be skipped without loss of continuity. Initially, our model involved constant coefficients rather than a neural network. We eventually found that the added flexibility of a neural network improved the performance of the numerical method. Initially, the neural network simply took the local solution values as the input and returned FVM coefficients as the output. Adding the affine transformation that guarantees consistency was found to significantly improve the performance of our numerical method. We initially enforced consistency by setting the neural network to output only four of the five coefficients and choosing the last coefficient such that the method is consistent. We ultimately changed this constraint to the optimization framework seen in this paper, as it seemed more elegant. However, this change had a very little effect on performance. Using the neural network to perturb the WENO5-JS coefficients was found to significantly improve performance over having the neural network directly output the coefficients. When the neural network directly outputs the coefficients, the trained numerical method is empirically unstable. To fix this issue, we applied L_2 regularization to the coefficients to add damping by biasing the coefficients to take on similar values. We then realized that we could instead bias these coefficients toward a scheme that we already know is stable, and modified the network architecture to perturb the fifth-order, constant coefficient method that WENO5-JS converges to in the presence of smooth solutions. We then decided to have the neural network perturb the WENO5-JS coefficients, which we found greatly improved the performance of the method. We also found that using these coefficients as the input to the neural network instead of the local solution values offered further improvement. Finally, we also experimented with the network size. We found that past a certain point, increasing the depth of the network and the number of nodes per layer did not improve performance, even with optimized regularization parameters. In order to minimize computational cost, we chose the smallest network that offered maximum performance, as this network is small and further decreasing its size was found to rapidly harm performance while offering very little speedup.

4 Results

4.1 Advection equation

These results will focus on comparing WENO5-JS to WENO-NN. Every WENO-NN result we show in this paper was generated using the same neural network with the same weights. As such, our numerical method

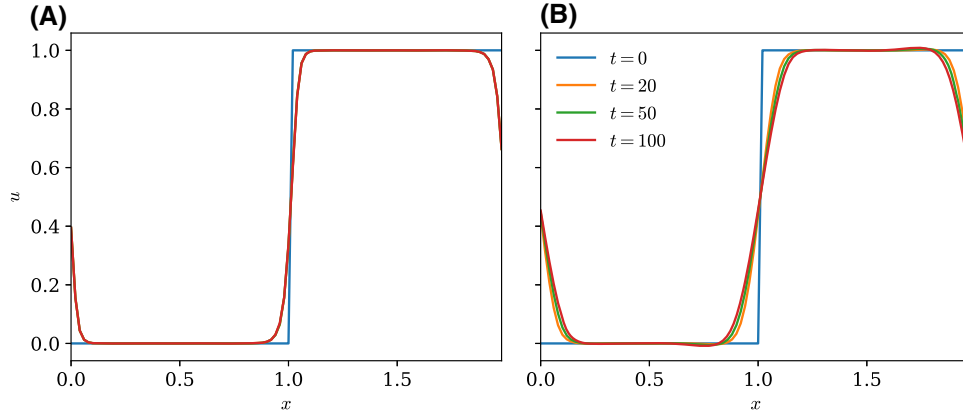


Fig. 3 Numerical solutions of the advection equation at $t = 0, 20, 50,$ and 100 using **a** WENO-NN and **b** WENO5-JS. Note that the curves in **a** for $t > 0$ are indistinguishable

is broadly applicable to problems not discussed in this paper, in contrast to many machine learning solutions that are problem specific. No additional training is necessary to use this method for other PDEs. The first test case we look at is the linear advection of a step function on a periodic domain. Mathematically, this IBVP is posed as

$$\begin{aligned} \frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} &= 0, \\ u(0, x) &= \begin{cases} 1, & \text{if } x \geq L/2, \\ 0, & \text{otherwise,} \end{cases} \\ u(t, 0) &= u(t, L). \end{aligned} \tag{13}$$

For this simulation, we set $c = 1$ and $L = 2$. We split the domain into 100 cells, use a CFL number of $2/3$, and run the simulation for 50 periods for a total time of $T = 100$. Figure 3 shows the solution of this PDE for WENO5-JS and WENO-NN at $t = 0, 20, 50,$ and 100 . The solution at $t = 0$ is also the exact solution at all the other times plotted.

One can see that the solution using WENO-NN provides a closer visual fit to the exact solution, as WENO5-JS diffuses the discontinuity more significantly than WENO-NN. WENO5-JS also introduces noticeable overshoot behind the discontinuity. The neural network has the interesting property that the waveform is nearly invariant to its propagation, while WENO5-JS continues to diffuse the solution. This behavior can be explained by examining the artificial fluid properties associated with the modified equation obtained by Taylor series expansion (assuming linearity of the scheme). The modified PDE is

$$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = v \frac{\partial^2 u}{\partial x^2} + \delta \frac{\partial^3 u}{\partial x^3} - \sigma \frac{\partial^4 u}{\partial x^4} + \dots \tag{14}$$

The expansions give expressions for the artificial viscosity, dispersion, and hyperviscosity, $\frac{\partial \bar{u}}{\partial t} + \frac{u(x + \frac{\Delta x}{2}) - u(x - \frac{\Delta x}{2})}{\Delta x} = 0$ after making the substitutions $u(x + \frac{\Delta x}{2}) = \sum_{n=-2}^2 c_n \bar{u}(x + n\Delta x)$ and $u(x - \frac{\Delta x}{2}) = \sum_{n=-2}^2 c_n \bar{u}(x + (n - 1)\Delta x)$, and are computed as

$$v = \Delta x \sum_{n=-2}^2 c_n \frac{(n-1)^2 - n^2}{2}, \tag{15}$$

$$\delta = \Delta x^2 \sum_{n=-2}^2 c_n \frac{(n-1)^3 - n^3}{6}, \tag{16}$$

$$\sigma = -\Delta x^3 \sum_{n=-2}^2 c_n \frac{(n-1)^4 - n^4}{24}. \tag{17}$$

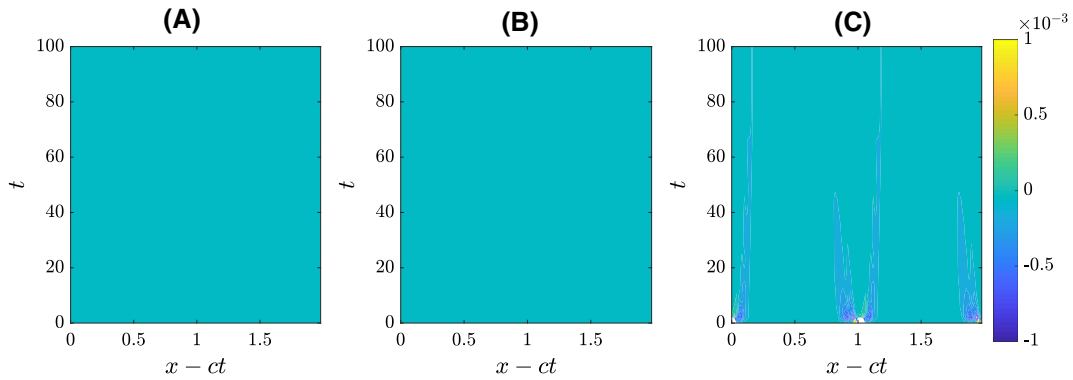


Fig. 4 Influence of **a** artificial viscosity, **b** dispersion, and **(C)** hyperviscosity of WENO5-JS

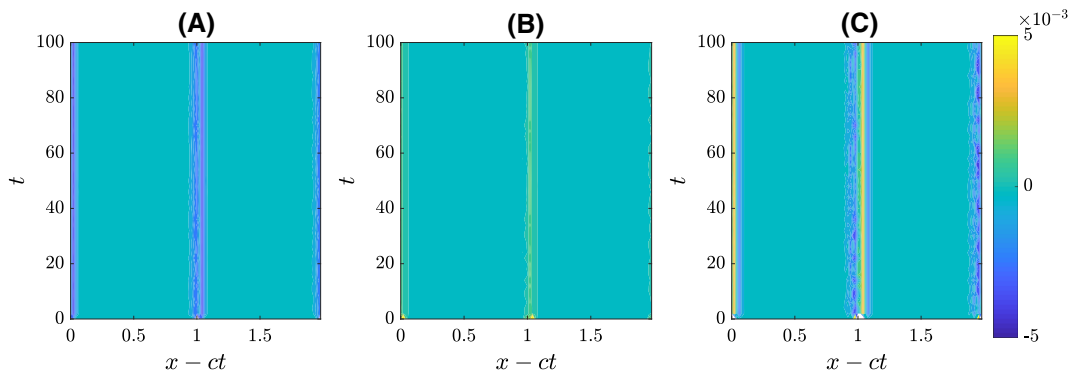


Fig. 5 Influence of **a** artificial viscosity, **b** dispersion, and **c** hyperviscosity of WENO-NN

Figure 4 shows these quantities for WENO5-JS. In order to estimate the contribution of each term, we approximated the higher-order spatial derivatives using standard finite-volume methods, and scale each by the magnitude of that derivative. For example, the influence of artificial viscosity is computed as

$$I_v(x) = \frac{v(x + \Delta x/2) + v(x - \Delta x/2)}{2} \left| \frac{u'(x + \Delta x/2) - u'(x - \Delta x/2)}{\Delta x} \right|. \quad (18)$$

Hence, we ignore regions of the flow where the coefficient may signify that artificial viscosity (or other properties) is being added when they would have a negligible effect because the derivative is small.

One can see that for WENO5-JS there is no viscosity or dispersion, as the method is designed such that on each substencil $\sum_{n=-2}^2 c_n \frac{(n-1)^2 - n^2}{2} = 0$ and $\sum_{n=-2}^2 c_n \frac{(n-1)^3 - n^3}{6} = 0$, so WENO5-JS applies only hyperviscosity. The method applies a small amount of negative hyperviscosity near the discontinuity. As time goes on and the discontinuity continues to diffuse, the influence of hyperviscosity decreases.

Figure 5 shows that unlike WENO5-JS, WENO-NN adds both artificial viscosity and dispersion to the solution. We see that near the discontinuity, negative viscosity is being added, which apparently provides the anti-diffusion that causes the discontinuity to retain its steepness, while hyperviscosity is applied to stabilize the solution.

We obtain a quantitative picture of the error in Fig. 6. We plot the L_2 error over time (measured to the exact solution), as well as the total variation, $TV = \sum_{i=1}^N |u(\Delta x i) - u(\Delta x(i-1))|$, to indicate when oscillations have been induced in the solution. We also measured the width over which the discontinuity is spread by counting the cells that have an error above a certain threshold (in this case chosen to be 0.01) and multiplying this number by $\Delta x/2$ since there are two discontinuities in the simulation.

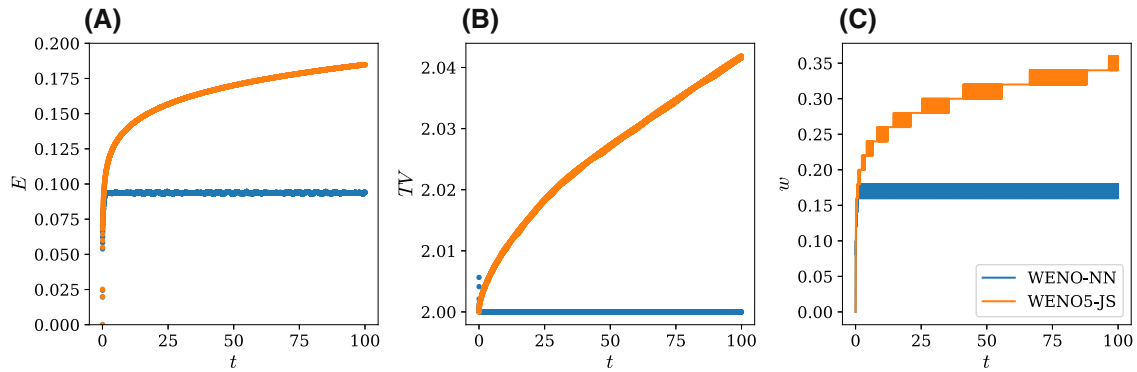


Fig. 6 Comparing **a** L_2 error, **b** total variation, and **c** discontinuity width over time for WENO-NN and WENO5-JS

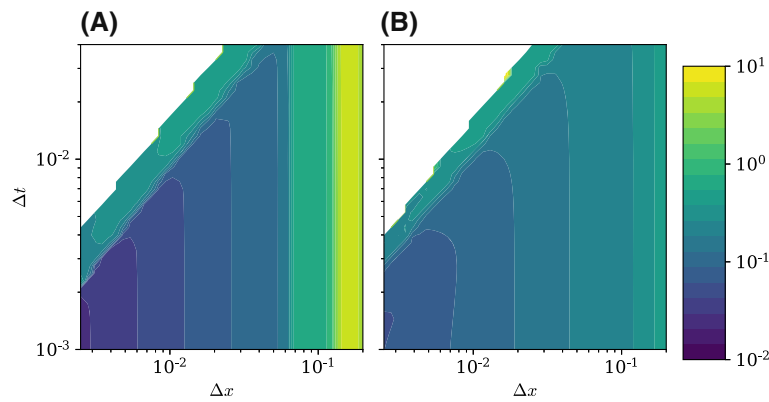


Fig. 7 L_2 error at the end of the simulation for **a** WENO-NN and **b** WENO5-JS

The figure shows that WENO-NN decreases the error by almost a factor of 2.¹ Although the total variation spikes at the start of the WENO-NN simulation, it is damped out and returns back to approximately the true value of 2, while the WENO5-JS total variation steadily climbs to above 2.04. We see a similar behavior in the discontinuity width, where WENO-NN reaches its steady value relatively quickly, while WENO5-JS continues to spread.

In order to determine how WENO-NN performs in different settings, the spatial and temporal discretizations were varied, and the L_2 error at the end of the simulation was measured. We again use a domain of length 2 and simulate for 50 periods. These results are shown in Fig. 7.

We can see that WENO-NN tends to outperform WENO5-JS in regions where the spatial discretization is fine, but results in a larger L_2 error for coarse discretizations. To further compare the methods, Fig. 8 shows the error against the run time for the two methods within a range of CFL values. We will only look at moderate CFL numbers, between 0.25 and 0.75, as stability becomes a concern for both methods above this range, and it becomes inefficient to run the simulation with CFL numbers below this range. We will also restrict the cell width to be below 0.025, as coarser meshes cause the final waveform to be unrecognizable compared to the exact solution for both methods, so the error comparison becomes meaningless. We see that when the CFL number is of a moderate value and the grid is sufficiently refined, WENO-NN typically achieves lower errors with smaller run time than WENO5-JS.

We also examine the convergence of each method for this problem in Fig. 9. Here, we fix $CFL = 0.5$ and measure the L_1 error of each numerical solution. We find that WENO1 achieves a slope of 0.5, WENO5-JS achieves a slope of 0.82, and WENO-NN achieves a slope of 1. Despite having a lower order of accuracy for smooth problems, WENO-NN is able to achieve a faster convergence rate for this discontinuous problem.

¹ Note that the error oscillates between two different values because in the exact solution the discontinuity switches between being on the edge of a cell and 1/3 of a cell width away from either the left or right of a cell edge since the CFL number is 2/3. To get a smooth curve, we apply a filter to the error and plot $E(i) = \frac{e(i)+e(i-1)+e(i-2)}{3}$.

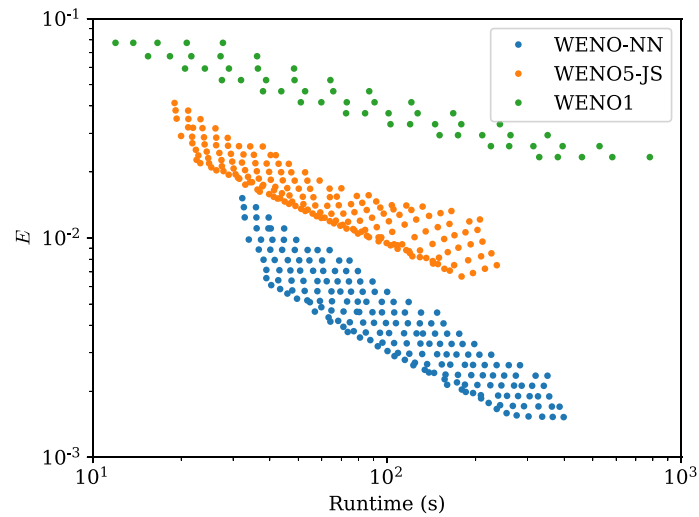


Fig. 8 Comparing the L_2^2 error and run time of WENO-NN, WENO5-JS, and WENO1 for $0.25 < CFL < 0.75$ and $\Delta x < 0.025$

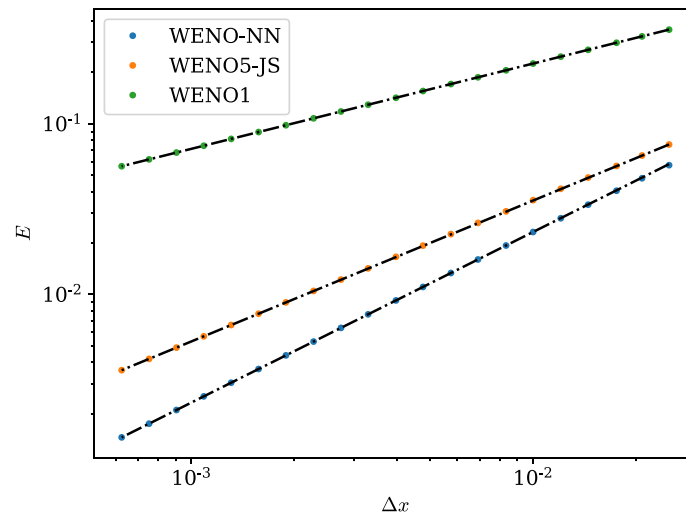


Fig. 9 Comparing the convergence rates of WENO-NN, WENO5-JS, and WENO1 for advection of a step function

4.2 Inviscid Burgers' equation

We next consider the inviscid Burgers' equation. Unlike the linear advection equation that included only contact (initial) discontinuities, the inviscid Burgers' equation results in shocks from smooth initial data. The distinction here is important: For a shock, the dynamics of the PDE will drive the solution toward a discontinuity, countering any diffusive effects associated with the numerics. We will again consider periodic boundary conditions, though we will start the simulation with a Gaussian as the initial condition. Hence, the IBVP is posed as

$$\frac{\partial u}{\partial t} + \frac{1}{2} \frac{\partial u^2}{\partial x} = 0, \quad (19)$$

$$u(0, x) = \exp\left(-k \left(x - \frac{L}{2}\right)^2\right), \quad (20)$$

$$u(t, 0) = u(t, L). \quad (21)$$

We simulate the problem for a time of $T = 4$ on a domain of length $L = 2$ and a value of $k = 20$. We first approximate the exact solution by solving this simulation with $\Delta x = 3.125 \cdot 10^{-4}$ and $\Delta t = 1.5625 \cdot 10^{-4}$ for

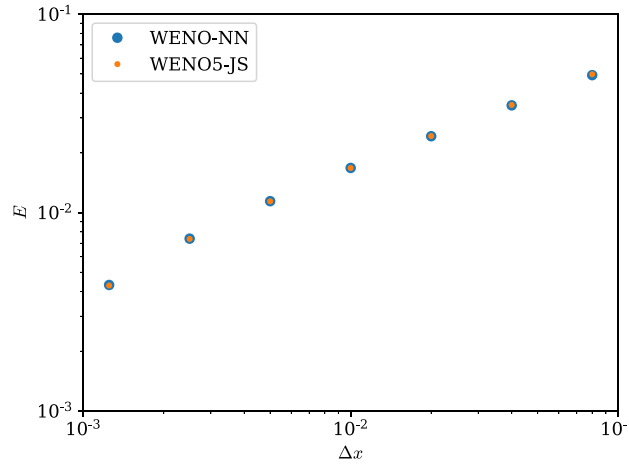


Fig. 10 Comparing error versus grid spacing of WENO-NN and WENO5-JS for the inviscid Burgers’ equation

a total of 6400 cells and 25,601 timesteps. What we see is that the L_2 error is roughly the same for WENO5-JS and WENO-NN, as shown in Fig. 10. Hence, we should expect the method to perform similarly to WENO5 in the presence of a shock.

4.3 1-D Euler equations

The last test case we will look at is the Shu–Osher problem, a test case involving the 1-D Euler equations. Note that the method was also tested on the Sod problem, but because this test case did not lead to any conclusions not drawn from either the advection equation or the inviscid Burgers’ equation, these results have been omitted. The Shu–Osher problem is a model problem for turbulence–shock wave interactions. It involves the following equations and initial conditions:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} = 0, \tag{22}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial(P + \rho u^2)}{\partial x} = 0, \tag{23}$$

$$\frac{\partial E}{\partial t} + \frac{\partial((E + P)u)}{\partial x} = 0, \tag{24}$$

$$P = (\gamma - 1) \left(E - \frac{1}{2} \rho u^2 \right), \tag{25}$$

$$\rho(0, x) = \begin{cases} 3.857143, & \text{if } x \leq 1 \\ 1 + \epsilon \sin(5x), & \text{otherwise} \end{cases}, \tag{26}$$

$$u(0, x) = \begin{cases} 2.629369, & \text{if } x \leq 1 \\ 0, & \text{otherwise} \end{cases}, \tag{27}$$

$$P(0, x) = \begin{cases} 10.33333, & \text{if } x \leq 1 \\ 1, & \text{otherwise} \end{cases}. \tag{28}$$

The simulation takes place on a domain of length $L = 10$ and is run until a final time of $T = 2$. ϵ is set to 0.2. We first obtain an approximately exact solution by discretizing the solution into 12,800 cells and 10,240 timesteps and use WENO5-JS for the simulation. This grid is fine enough to consider the solution exact. We then solve the problem using 300 cells and 240 timesteps using both WENO5-JS and WENO-NN and compare the numerical results to the exact solution. Figure 11 shows the density, pressure, and velocity at the end of the simulation.

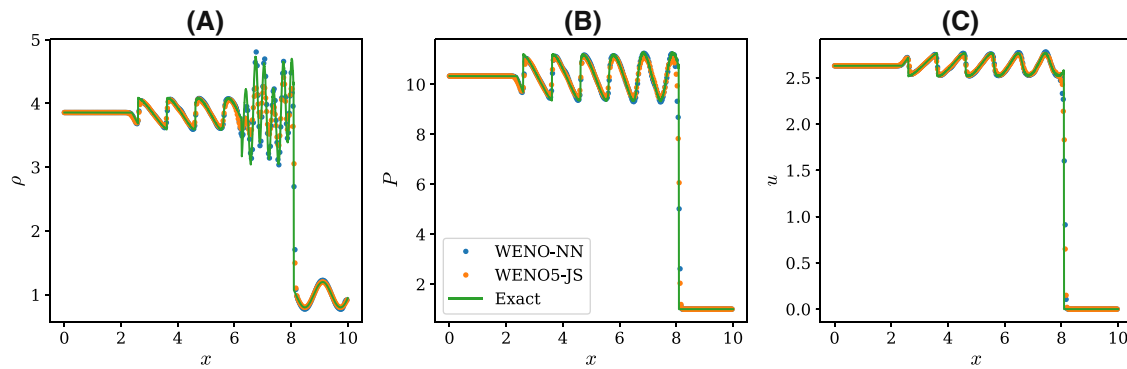


Fig. 11 Comparing **a** density, **b** pressure, and **c** velocity of WENO-NN and WENO5-JS to the exact solution for the Shu–Osher problem

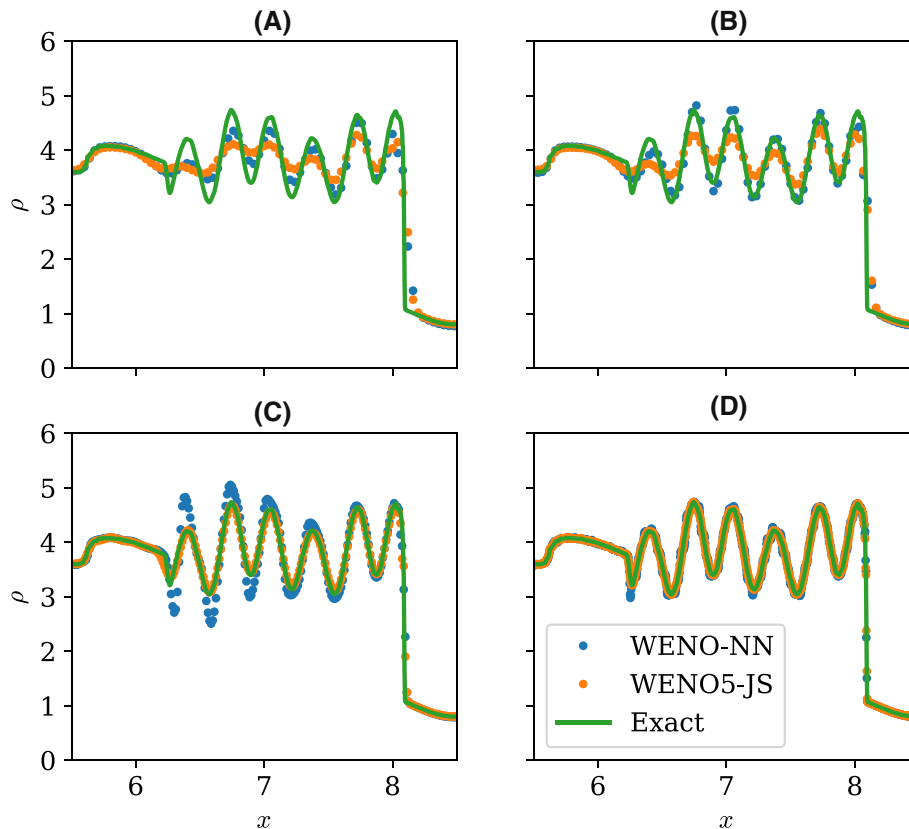


Fig. 12 Zoomed-in view of the turbulent section for different grid resolutions of **a** 250, **b** 300, **c** 800, and **d** 3200 cells for the Shu–Osher problem

The most interesting aspect of the solution is the highly oscillatory section of the density profile, which is considered to behave similarly to turbulence. Figure 12 shows a zoomed-in view of this section at different grid resolutions.

One can see that the neural network diffuses the oscillations significantly less than WENO5 for coarse grids, which is an encouraging result in terms of simulating actual turbulence. As the mesh is further refined, the WENO-NN appears to add too much anti-diffusion, which inflates fine features of the solution. On the very fine grid, both WENO5-JS and WENO-NN are similar (provided WENO-NN is stable, and then it is constrained to converge as at least first order).

5 Discussion and conclusions

By training a neural network to process the outputs of the WENO5-JS algorithm, we were able to improve its accuracy, particularly in problems where the artificial diffusion introduced in WENO5-JS was excessive. While WENO-NN is more expensive per evaluation than WENO5-JS, it achieved lower errors on coarser grids, which indicates some potential to be useful more generally. We trace these performance improvements to increased flexibility in the neural network compared to WENO5-JS, as it can add artificial viscosity and dispersion, while WENO5-JS coefficients are constrained to make these quantities zero. By analyzing the advection of a step function, we found that WENO-NN applies negative artificial viscosity near the discontinuity, which allows it to maintain its sharp profile (this takes place sometime into the simulation after the initial profile has been slightly smoothed due to artificial viscosity that prevents spurious oscillations). We then observe similar behavior in the Shu–Osher problem, where we see that WENO5-JS diffuses the fine features of the solution more than WENO-NN. However, we also found that at certain resolutions WENO-NN applies too much negative artificial viscosity, resulting in too much amplification of these fine-scale features, though this amplification does not develop into an instability. For true shocks, as opposed to contact discontinuities, we found that our method performs very similarly to WENO5-JS.

One drawback of WENO-NN is that it does not inherit the high-order convergence of WENO5-JS. It would be an improvement to the method to be able to structure the network such that its coefficients more quickly converge to those of either WENO5-JS or the constant coefficient scheme that maximizes order of accuracy in the presence of smooth solutions. However, this must be done in a way that does not interfere with predictions in non-smooth regimes that benefit from low-order behavior, which is a non-trivial task. Until such a method is developed, one would need to use WENO-NN as part of a hybrid scheme if higher-order convergence is desired in smooth regions [20]. Another outstanding issue with machine-learned schemes is stability. The WENO-NN scheme used here seemed to inherit the stability of the underlying WENO5-JS scheme that it was based on, but this need not have been the case, and we cannot offer proof or an estimate for the maximal CFL.

In future work, we aim to test the method on large-scale, multidimensional problems. We would expect the benefits seen in 1-D problems to be more significant when multiple spatial dimensions are present, as WENO-NN allows for a coarser mesh, so the improvement scales exponentially with the number of dimensions.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Bar-Sinai, Y., et al.: Learning data-driven discretizations for partial differential equations. *Proc. Natl. Acad. Sci.* **116**(31), 15344–15349 (2019)
2. Borges, R., et al.: An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws. *J. Comput. Phys.* **227**(6), 3191–3211 (2008)
3. Castro, M., Costa, B., Don, W.S.: High order weighted essentially non-oscillatory WENO-Z schemes for hyperbolic conservation laws. *J. Comput. Phys.* **230**(5), 1766–1792 (2011)
4. Chollet, F. et al.: Keras. <https://keras.io> (2015)
5. Chu, C.K.: Numerical methods in fluid dynamics. In: *Advances in Applied Mechanics*, vol. 18. Elsevier, Amsterdam, pp. 285–331 (1979)
6. Dissanayake, M.W.M.G., Phan-Thien, N.: Neural-network-based approximations for solving partial differential equations. *Commun. Numer. Methods Eng.* **10**(3), 195–201 (1994)
7. Fang, J., Li, Z., Lu, L.: An optimized low-dissipation monotonicity preserving scheme for numerical simulations of high-speed turbulent flows. *J. Sci. Comput.* **56**(1), 67–95 (2013)
8. Gottlieb, S., Shu, C.W.: Total variation diminishing Runge–Kutta schemes. *Math. Comput. Am. Math. Soc.* **67**(221), 73–85 (1998)
9. Ha, Y., et al.: An improved weighted essentially non-oscillatory scheme with a new smoothness indicator. *J. Comput. Phys.* **232**(1), 68–86 (2013)
10. Harten, A.: High resolution schemes for hyperbolic conservation laws. *J. Comput. Phys.* **49**(3), 357–393 (1983)
11. Harten, A. et al.: Uniformly high order accurate essentially non-oscillatory schemes, III. In: *Upwind and High-Resolution Schemes*. Springer, Berlin, pp. 218–290 (1987)
12. Hsieh, J.T. et al.: Learning neural PDE solvers with convergence guarantees. In: arXiv preprint [arXiv:1906.01200](https://arxiv.org/abs/1906.01200) (2019)
13. Jiang, G.S., Shu, C.W.: Efficient implementation of weighted ENO schemes. *J. Comput. Phys.* **126**(1), 202–228 (1996)

14. Kim, C.H., Ha, Y., Yoon, J.: Modified non-linear weights for fifth-order weighted essentially non-oscillatory schemes. *J. Sci. Comput.* **67**(1), 299–323 (2016)
15. Kim, J.W., Lee, D.J.: Optimized compact finite difference schemes with maximum resolution. *AIAA J.* **34**(5), 887–893 (1996)
16. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014)
17. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **9**(5), 987–1000 (1998)
18. Lele, S.K.: Compact finite difference schemes with spectral-like resolution. *J. Comput. Phys.* **103**(1), 16–42 (1992)
19. LeVeque, R.J., et al.: *Finite Volume Methods For Hyperbolic Problems*, vol. 31. Cambridge University Press, Cambridge (2002)
20. Li, G., Qiu, J.: Hybrid weighted essentially non-oscillatory schemes with different indicators. *J. Comput. Phys.* **229**(21), 8105–8129 (2010)
21. Liu, Y.: Globally optimal finite-difference schemes based on least squares. *Geophysics* **78**(4), T113–T132 (2013)
22. Peer, A., Dauhoo, M.Z., Bhuruth, M.: A method for improving the performance of the WENO5 scheme near discontinuities. *Appl. Math. Lett.* **22**(11), 1730–1733 (2009)
23. Pfau, D. et al.: Spectral inference networks: unifying deep and spectral learning. In: arXiv preprint [arXiv:1806.02215](https://arxiv.org/abs/1806.02215) (2018)
24. Rathan, S., Raju, G.N.: A modified fifth-order WENO scheme for hyperbolic conservation laws. *Comput. Math. Appl.* **75**(5), 1531–1549 (2018)
25. Rathan, S., Raju, G.N.: Improved weighted ENO scheme based on parameters involved in nonlinear weights. *Appl. Math. Comput.* **331**, 120–129 (2018)
26. Shu, C.W.: Essentially non-oscillatory and weighted essentially nonoscillatory schemes for hyperbolic conservation laws. In: *Advanced numerical approximation of nonlinear hyperbolic equations*. Springer, Berlin, pp. 325–432 (1998)
27. Shu, C.W.: High-order finite difference and finite volume WENO schemes and discontinuous Galerkin methods for CFD. *Int. J. Comput. Fluid Dyn.* **17**(2), 107–118 (2003)
28. Tam, C.K., Webb, J.C.: Dispersion-relation-preserving finite difference schemes for computational acoustics. *J. Comput. Phys.* **107**(2), 262–281 (1993)
29. Wang, Z.J., Chen, R.F.: Optimized weighted essentially nonoscillatory schemes for linear waves with discontinuity. *J. Comput. Phys.* **174**(1), 381–404 (2001)
30. Yu, J., Hesthaven, J.S., Yan, C.: A data-driven shock capturing approach for discontinuous Galerkin methods. Tech. rep. (2018)
31. Zhang, J.H., Yao, Z.X.: Optimized explicit finite-difference schemes for spatial derivatives using maximum norm. *J. Comput. Phys.* **250**, 511–526 (2013)