mathematics

MDPI

Article

Coarse-Gridded Simulation of the Nonlinear Schrödinger Equation with Machine Learning

Benjamin F. Akers * and Kristina O. F. Williams

Department of Mathematics, Air Force Institute of Technology, 2950 Hobson Way, Wright-Patterson AFB, Dayton, OH 45433, USA; kristina.williams.4@us.af.mil
* Correspondence: benjamin.akers@us.af.mil

Abstract: A numerical method for evolving the nonlinear Schrödinger equation on a coarse spatial grid is developed. This trains a neural network to generate the optimal stencil weights to discretize the second derivative of solutions to the nonlinear Schrödinger equation. The neural network is embedded in a symmetric matrix to control the scheme's eigenvalues, ensuring stability. The machine-learned method can outperform both its parent finite difference method and a Fourier spectral method. The trained scheme has the same asymptotic operation cost as its parent finite difference method after training. Unlike traditional methods, the performance depends on how close the initial data are to the training set.

Keywords: machine learning; nonlinear Schrödinger equation; coarse grid

MSC: 65M20; 68T05; 74J30

1. Introduction

A numerical time-stepping algorithm is developed to approximate the solution of a partial differential equation—the cubic nonlinear Schrödinger equation (NLS) in the coarse discretization regime. Recently, machine learning-based methods have been proposed as a method for creating PDE solvers when classic methods are under-resolved [1–4]. In this work, a machine learning approach is developed to create a numerical scheme for solving the nonlinear Schrödinger equation (NLS). The method uses a neural network to generate spatial differencing weights in an approximation of the second derivative. The network is trained using the truncation error of the discretization (using a second-order implicit-explicit [IMEX2] procedure for temporal stepping) as an objective function. The training set is generated using a Fourier spectral method on a fine grid, which is then downsampled to a coarse grid before training (*the method could also be trained on exact solutions*). The result is a numerical scheme which can outperform both Fourier collocation and its parent finite difference scheme when used on coarse grids and on initial data near the training set.

The equation simulated in this work is the focusing-type cubic nonlinear Schrödinger equation (NLS). NLS is a common model used when the envelope of a wave packet is a quantity of interest [5–9]. NLS is fully integrable [10], can be solved using the inverse scattering transform [11], and contains solitary wave solutions. None of these properties are required for the machine-learned algorithm presented here. NLS was chosen as it requires the extension of [1] to a complex-valued solution in an equation with a symmetric linear operator and for its importance in laser propagation [9,12]. The resulting scheme generalizes to other equations where symmetric differential operators are discretized, for example, Poisson, Navier–Stokes, and the heat equation. These other equations arise in important applications, such as numerical weather prediction, computational fluid dynamics, and aerospace [13]. The nonlinear Schrödinger equation [14] takes the form

$$iu_t + u_{xx} + 2|u|^2 u = 0. \tag{1}$$

check for updates

Citation: Akers, B.F.; Williams, K.O.F. Coarse-Gridded Simulation of the Nonlinear Schrödinger Equation with Machine Learning. *Mathematics* **2024**, *12*, 2784. https://doi.org/10.3390/math12172784

Academic Editors: Thái Anh Nhan and Istvan Farago

Received: 1 August 2024
Revised: 30 August 2024
Accepted: 6 September 2024
Published: 9 September 2024

This is the focusing case of (1), which has closed-form solitary wave solutions

$$\tilde{u}(x,t) = \alpha \text{sech}(\alpha(x - 2\beta t)) \exp\Big(i(\beta x + (\alpha^2 - \beta^2)t)\Big), \tag{2}$$

where α is the amplitude and β is the speed [14].

Machine learning methods are a popular tool for studying PDEs [15–17]. They can be used to find the PDE underlying a set of data [18–23]. They have been used to improve the accuracy of solutions to PDEs [3,18,24–30]. Recently, neural networks have been used to create new PDE solvers [3,15,18,31–33]. These works use a variety of layer depths, ranging from 3 to 22 layers. To avoid the known dangers of deep networks, including long training time and/or overfitting [32,33], this work uses a single-layer fully connected neural network.

The machine learning model in this work serves as a spatial discretization of the second derivative term, with fixed stencil width (see [3] for an example with variable stencil). The spatial discretization is paired with an IMEX2 scheme for time-stepping. IMEX2 was chosen for its unbounded stability region [34]. Fourth-order Runge–Kutta (RK4) [18,26,28,29,35] is a popular competitor as a time-stepping algorithm, but the finite stability region of explicit methods poses a stability risk from a trained spatial differencer. In addition to choosing a time-stepper with an unbounded stability region, the machine learning network is built in such a way as to preserve the spatial operator's symmetry, guaranteeing eigenvalues which live in the stability region of the scheme. The resulting scheme is stable for all time regardless of spatial or temporal grid spacing, a challenge for machine-learned time-steppers, as noted in [3]. This novel approach both gives a stable scheme for the nonlinear Schrödinger equation in this work, but generalizes to other partial differential equations with symmetric differential operators (Poisson, heat equation, Navier–Stokes, etc.).

The machine-learned scheme is tested on a family of initial data and compared in performance against a Fourier spectral and second-order finite difference discretization. The machine-learned scheme gives smaller errors than either of these methods for data sufficiently close to the training set but loses accuracy when the initial data are far from the training set.

There are many known strategies employed for creating training and test data. For applications in PDEs, training data can be generated from highly resolved classic methods (e.g., finite difference or Fourier spectral) [18,19,21,25,26,28,36]. A single short time interval can be used for training [1], or a random ensemble of long time data can be used for training [3]. In this work, a single solution trajectory is used for training, and testing is conducted on the time dynamics of solutions with nearby initial data.

The remainder of the paper is organized as follows: Section 2 discusses the machine learning model, the numerical scheme, and the training procedure. Section 3 presents the results of the training scheme and comparisons against classic methods, including both cases where the machine learning method outperforms, and those where it is outperformed by, its competition. Section 4 provides conclusions and avenues for future research.

2. Methodology

The machine learning method uses a neural network to improve a second-order centered finite difference scheme approximation of the second derivative,

$$u_{xx} = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + O(h^2), \tag{3}$$

and IMEX2 time-stepper,

$$\frac{u^{n+1} - u^{n-1}}{2k} = \frac{1}{2}\Big(g(u^{n+1}) + g(u^{n-1})\Big) + f(u^n). \tag{4}$$

In (4), the function $g(u)$ is the discretization of the linear term in Equation (1) and $f(u)$ a discretization of the cubic nonlinearity. Two discretizations of the nonlinearity are considered,

$$f_j^{(1)}(\mathbf{u}) = |u_j|^2 u_j, \text{ and } f_j^{(2)}(\mathbf{u}) = |u|_j^2 (u_{j+1} + u_{j-1})/2.$$

The latter has been shown to outperform the former on coarse grids [37].

This scheme has the linear stability region of the trapezoid rule, and thus is unconditionally stable when the spectrum of the linear operator is pure imaginary. While the scheme is stable for any time and space step pair, very coarse grids make the scheme useless due to large truncation errors. The goal of the machine-learned scheme developed in this work is to extend the range over which the scheme is useful, by decreasing the truncation error for large spatial step sizes.

The machine-learned model replaces the stencil weights of the centered difference approximation (3) with a neural network. The neural network stencil weights are restricted to preserve the symmetry of the operator, choosing the coefficients on the subdiagonal of the differencing matrix to equal those on the superdiagonal. The weights a_1 (on the superdiagonal of the differencing matrix) and a_2 (on the diagonal of the differencing matrix) are functions of u, defined as

$$\mathbf{a}(\mathbf{u}^n)_j = \mathbf{b} + \mathbf{W}_1 \sigma \left(\mathbf{W}_2 \begin{pmatrix} \text{real}(\mathbf{X}_j \mathbf{u}^n) \\ \text{imag}(\mathbf{X}_j \mathbf{u}^n) \end{pmatrix} \right). \tag{5}$$

The vector $\mathbf{b}$ and matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ contain the training parameters. Their dimensions are $b \in \mathbb{R}^2$, $W_1 \in \mathbb{R}^{2\times 2}$, and $W_2 \in \mathbb{R}^{2\times 6}$. The function σ is an activation function, where $\sigma(x) = \tanh(x)$, which acts on the input vector component-wise. No hyperparameters were used. The entire network is explicit in (5), which is fully connected, the matrices W_1 and W_2 matrices are full, and contains a single activation function. $\mathbf{X}_j$ is a $3 \times N$ matrix with entries

$$(\mathbf{X}_j)_{i,l} = \delta_{i,(l-j+2) \bmod N},$$

which is used to restrict spatial vectors to the three-point stencil centered at point x_j. The modulo N in the second index imposes periodicity in the spatial dimension. The result is that $\mathbf{X}_j$ applied to u^n restricts the support of the input vector to its three-point stencil,

$$\mathbf{X}_j \mathbf{u}^n = \begin{pmatrix} u_{j-1}^n \\ u_j^n \\ u_{j+1}^n \end{pmatrix}. \tag{6}$$

Equation (6) is presented for $j \in [2, N-1]$; for $j = 1$ or N, the periodicity in the j-index will be evident (and is explicit in the definition of X_j).

The outputs of the machine-learned model are stencil weights for the approximation of the second derivative, of the form $\mathbf{u}_{xx} \approx \mathbf{B}(\mathbf{u})\mathbf{u}$, with

$$\mathbf{B}(u) = \begin{bmatrix} (a_2)_1 & (a_1)_1 & 0 & \dots & & \dots & 0 & (a_1)_N \\ (a_1)_1 & (a_2)_2 & (a_1)_2 & 0 & \dots & & \dots & 0 \\ 0 & (a_1)_2 & \ddots & \ddots & 0 & \dots & & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 & \dots & \\ & \vdots & 0 & \ddots & \ddots & \ddots & 0 & \dots \\ & & \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \vdots & 0 & \ddots & (a_2)_{N-2} & (a_1)_{N-2} \\ 0 & \vdots & & & \vdots & 0 & (a_1)_{N-2} & (a_2)_{N-1} \\ (a_1)_N & 0 & \dots & & & \vdots & 0 & (a_1)_{N-1} \end{bmatrix} \begin{matrix} (a_1)_N \\ 0 \\ \vdots \\ \\ \\ \\ (a_1)_{N-1} \\ (a_2)_N \end{matrix}.$$

Indicially, the second derivative operator is approximated

$$(u_{xx})_j \approx \frac{(a_1)_{j-1}u_{j-1} + (a_2)_j u_j + (a_1)_j u_{j+1}}{h^2}. \tag{7}$$

Notice that the weight on u_{j-1} is $(a_1)_{j-1}$, rather than $(a_1)_j$. With this choice, the second derivative matrix is symmetric. The symmetry of the matrix, $\mathbf{B}(\mathbf{u})$, ensures that this machine-learned operator's spectrum lies in the stability region of IMEX2 regardless of the training results.

The combined time-stepping scheme to be trained is

$$i\frac{\mathbf{u}^{n+1} - \mathbf{u}^{n-1}}{2k} = -\frac{1}{2h^2}\mathbf{B}(\mathbf{u}^n)\mathbf{u}^{n+1} - \frac{1}{2h^2}\mathbf{B}(\mathbf{u}^n)\mathbf{u}^{n-1} + f(\mathbf{u}^n). \tag{8}$$

Strictly speaking, the above scheme is no longer the IMEX2 in (4), since the operator g now depends on u at two different time steps. The new scheme, (8), is still second order, and has the same linear stability region as (4). The scheme is not self-starting; IMEX1 is used for the first time step. In Equation (8) and throughout, j indexes a spatial location, k is the temporal step size, h is the spatial step size, and n indexes the spatial location.

The scheme was trained using the truncation error against a highly resolved numerical simulation, $\tilde{u}$, downsampled to the coarse grid as the objective function,

$$H(b, W_1, W_2) = \left\| i\frac{\tilde{u}^{n+1} - \tilde{u}^{n-1}}{2k} + \frac{1}{2}\mathbf{B}(\tilde{u}^n)\tilde{u}^{n+1} + \frac{1}{2h^2}\mathbf{B}(\tilde{u}^n)\tilde{u}^{n-1} - f(\tilde{u}^n) \right\|_F^2. \tag{9}$$

The argument of the norm is the scheme at some choice of space and time points, a doubly indexed set. The argument is interpreted as a matrix, so the objective function is the square of the Frobenius norm of that matrix. The size of the training set comes from the number of spatial and temporal points this objective function is evaluated on, both of which are training parameters. The objective function, H in Equation (9), is minimized with classic steepest descent. Training is considered successful only if the objective function is decreased by a factor of ten from its initialization. The unknowns are initialized with classic centered finite difference as an initial guess, $\vec{b} = [1, -2]^T$, $\mathbf{W}_1 = \mathbf{0}$, and $\mathbf{W}_2 = \mathbf{0}$.

In the testing phase of this procedure, both the truncation error (9) and the forward error are used. The forward error is calculated against highly resolved Fourier solutions, u_F, which are downsampled after their computation onto the coarse grid used by the machine-learned scheme. The forward error might be considered preferable as an objective function, but costs significantly more to evaluate, and we observe that the forward error and truncation error closely track one another.

In summation, a neural network is used for the weights in a differencing scheme (8). The training set is a coarse sampling of a highly resolved simulation (simulated on a fine grid) along a single solution trajectory. The testing set comprises other solution trajectories from different initial data. The machine-learned scheme has the same stencil breadth as the finite difference scheme. The memory and operational cost of using the machine-learned method after training are asymptotically identical to the finite difference, $O(n)$ flops per time step. The training procedure is a significant extra cost, but may be worth it when the trained method can achieve error thresholds which other methods cannot.

3. Results

The method was trained on a single solution trajectory, whose initial data are taken from

$$u(x, 0) = D[\alpha \operatorname{sech}(\alpha x) \exp(i\beta x)]. \tag{10}$$

For some choices of $\alpha, \beta, D \in \mathbb{R}$. When $D = 1$, this is a soliton solution of NLS [14] whose envelope travels as speed 2β, with formula

$$u(x,t) = \alpha \text{sech}(\alpha(x - 2\beta t)) \exp(i\beta x + i(\beta^2 - \alpha^2)t). \quad (11)$$

If $D \neq 1$, the initial data are a dilation of a solitary wave's and the solution does not correspond to a single NLS soliton but has some other dynamic (e.g., including dispersive radiation [11]).

The training data were generated from a finely discretized, highly resolved numerical simulation. This simulation used Fourier collocation for the spatial derivatives and a second-order IMEX time-stepper for time-stepping. The resolutions used to generate the training set were $N_x = 512$, $L = 40$, and $k = 0.002$. This same temporal step size was used for all simulations (training and testing), making the truncation error due to the temporal step size approximately four orders of magnitude smaller than the spatial truncation error. The highly resolved simulation was then downsampled to a coarse grid, with $N_x = 16$ and $h = 2.5$. The simulation time step was chosen so that the truncation errors in time are negligible compared to the truncation errors in space. The smallest truncation errors achieved by simulations on the coarse grid are $\approx 10^{-3}$. These are presumed to be almost entirely spatial based, since a second-order time-stepper is being used with a time step $k = 2 \times 10^{-3}$. All increases in accuracy of the method are attributed to the increased accuracy of the spatial discretization due to training on the coarse grid. All simulations were developed and conducted in MATLAB.

A single training set was used to find the neural network used to generate the results of this section. This network was trained on the trajectory of a single NLS traveling solitary wave, with $D = 1, \alpha = 0.75$ and $\beta = 0.2$ in Equation (11). The objective function for training was the truncation error (9), evaluated at 50 equally spaced times in the interval $0 \leq t \leq 4$. The objective was minimized using the steepest descent, until the truncation error reached a threshold of 1/10th of the truncation error of its parent finite difference method. The parent finite difference method takes $b = [1, -2]^T$, $\mathbf{W}_1 = \mathbf{0}$, and $\mathbf{W}_2 = \mathbf{0}$, and was used as an initial guess for the minimization procedure.

Because the method is trained using the truncation error as the objective function, the resulting neural network is guaranteed to have *truncation* error, which is $10\times$ smaller than its parent finite difference when evaluated on the training set. Since both schemes are linearly stable, one might expect that the *forward* error would also be $10\times$ smaller. Unfortunately, there is no such guarantee for the forward error; it is possible for one scheme to have truncation errors that cancel out rather than accumulate, even though the schemes have the same stability. Thus, the ordering of the truncation errors does not imply an ordering on the forward errors. Such cancellation was not observed in these simulations, and the achieved forward errors were $10\times$ smaller for the machine-learned scheme than the parent finite difference method on the training set.

A series of experiments were conducted to test the machine-learned numerical method. The simplest was comparison of solution profiles during the propagation of non-solitary initial data. The chosen initial data come from (10) with $D = 1.1$ ($D = 1$ would be solitary), $\beta = 0.22$, and $\alpha = 0.825$; all three parameters were changed by 10% from the trained values. The evolution is presented to $t = 6$ (50% longer than the training time interval). Figure 1 depicts the results. The machine-learned differencing method best approximates the exact solution at all times. It outperforms its parent finite difference method but also defeats Fourier collocation. Thus, for this resolution and initial data, the machine learning method is the superior scheme. The method is guaranteed to be outperform its parent for within the training interval. Longer time simulations are in Figure 2; at the simulated resolution eventually all three schemes have $O(1)$ error.

The accuracy of the machine-learned differencing scheme depends on the distance between the solution and the training set. This dependence was probed with a series of test simulations. In the examples presented here, the spatial discretization uses 16 points, so the initial data are in $\mathbb{C}^{16}$. Rather than attempt to visualize the regions close to the training set in sixteen dimensional space, cross-sections of the initial data space were taken, using the parameter values of D, α, and β from (10).

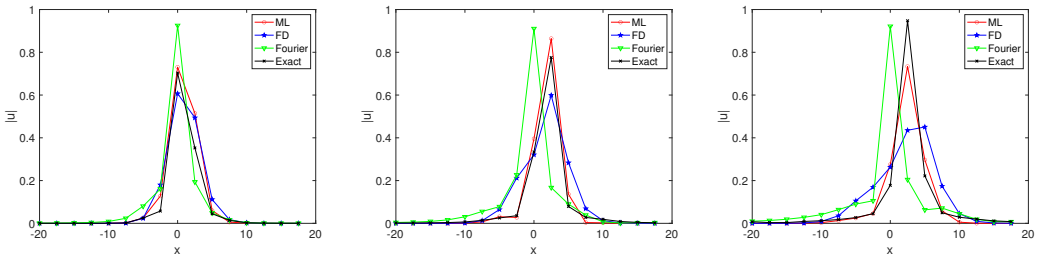

Figure 1. An example time evolution on a coarse grid ($N = 16$ points, $\Delta x = 2.5$) using the machine-learned time-stepper (in red marked by circles), the parent finite difference (in blue marked by stars), a Fourier pseudospectral collocation (in green marked by triangles), and the exact solution (in black marked by exes). The nonlinearity in the finite difference and machine-learned method both use the three-point stencil of [37]. The initial data for this simulation took $D = 1.1, \alpha = 0.825$, and $\beta = 0.22$ in (10), each 10% off of the training data for the machine-learned method whose trajectory was $D = 1, \alpha = 0.75$ and $\beta = 0.2$. The solution is depicted at times $t = 2, t = 4$, and $t = 6$; the training set included $0 \leq t \leq 4$.

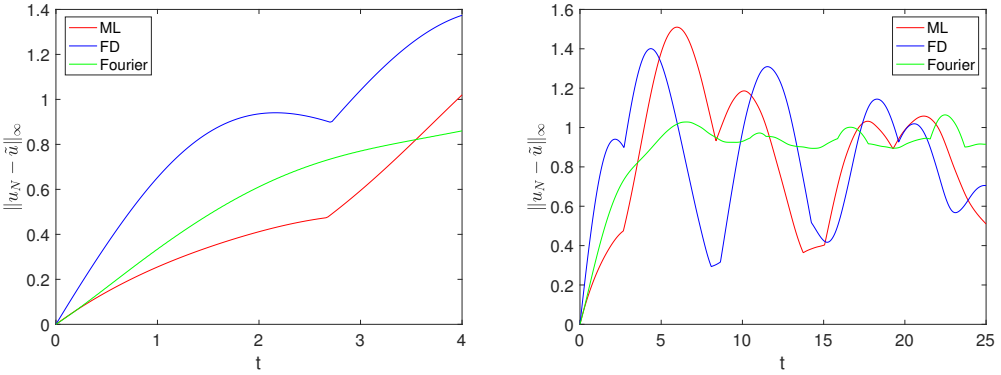

Figure 2. The L_∞ spatial error of numerical solutions u_N from each method in the same configuration as Figure 1 are plotted as a function of time. The machine-learned time-stepper is marked in red, the parent finite difference is in blue, and the Fourier pseudospectral collocation is in green. The nonlinearity in the finite difference and machine-learned method both use the three-point stencil of [37]. The **left panel** is over the training interval, where the machine-learned method is designed to outperform its parent finite difference. The **right panel** depicts the same error for a longer time, where all three methods ultimately have order one error.

The metric used to observe the performance of the machine-learned differencing scheme as the initial data get farther from the training set was the ratio of the forward error of the machine-learned method to the forward error of a competing method,

$$\text{Forward Error Ratio} = \frac{\|\tilde{u} - u_{ML}\|}{\|\tilde{u} - u_C\|}. \tag{12}$$

In Equation (12), u_{ML} is the solution generated by the machine-learned differencing scheme, $\tilde{u}$ is a solution from a highly resolved Fourier collocation method (a proxy for an exact solution), and u_C is the solution computed by a competing method (for example, the parent finite difference scheme or Fourier collocation conducted on a coarse grid).

In the first set of experiments, the machine-learned scheme is compared to its parent finite difference method. In all cases, the machine-learned scheme uses the weights for a classic centered difference scheme as an initial guess for training. Two different discretizations of the nonlinearity are compared. The first discretizes the nonlinearity at a single point and is known to have chaotic dynamics on coarse grids [37]. The second discretizes the nonlinearity at three points and is absent chaos. Figures 3 and 4 compare the relative forward errors of the parent finite difference schemes and the machine-learned children

at a variety of initial data, from (10). Figure 3 presents the relative accuracy of the parent and child schemes as functions of speed and amplitude for traveling solitary wave initial data. Both discretizations of the nonlinearity include regions near the training point where the machine-learned scheme has forward error, which is $10\times$ smaller than the parent finite difference scheme. Figure 3 presents the relative errors in the case when the initial data have non-trivial phase ($\beta = 0.2$) whose propagation is non-solitary ($D \neq 1$), varying in breadth and amplitude. Again both discretizations include regions near the training set where the forward error has decreased by a factor of 10.

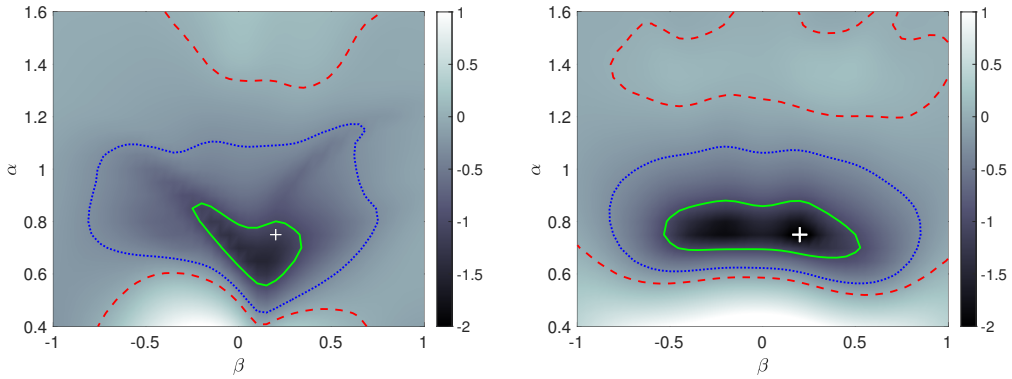

Figure 3. The ratio of the forward error of a classic finite difference to the forward error of the machine-learned differencing method is reported for a sampling of initial data, with $D = 1$ (soliton) varying α and β in Equation (10). The solid green contour marks when the machine-learned solution is $10\times$ more accurate; the dotted blue curve marks where the machine-learned solution is twice as accurate. The methods are equally accurate along the red dashed curve. The **left panel** discretizes the nonlinearity at a single point; the **right panel** discretizes the nonlinearity using the three-point stencil of [37]. The training data is marked with a plus sign.

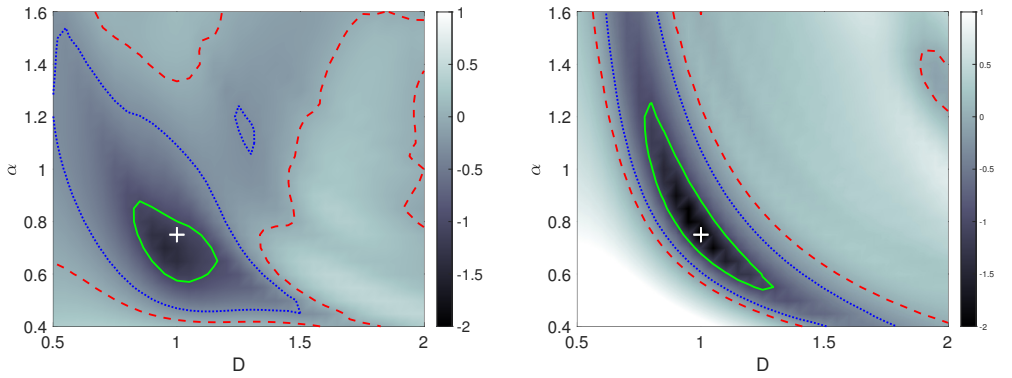

Figure 4. The ratio of the forward error of a classic finite difference to the forward error of the machine-learned differencing method is reported for a sampling of initial data, with $\beta = 0.2$ varying α and D in Equation (10). The solid green contour marks when the machine-learned solution is $10\times$ more accurate; the dotted blue curve marks where the machine-learned solution is twice as accurate. The methods are equally accurate along the red dashed curve. The **left panel** discretizes the nonlinearity at a single point, the **right panel** discretizes the nonlinearity using the three-point stencil of [37]. The training data is marked with a plus sign.

The machine-learned difference methods were trained via optimization beginning with the parent finite difference method as the initialization. Although not guaranteed, it is not surprising that machine-learned methods outperform their parents near the training set. In Figure 5, the relative accuracy of the machine-learned differencing scheme with the three-point nonlinear discretization of [37] is compared to the parent finite difference method (left panel) as well as to the Fourier collocation calculated on the same coarse grid

($\Delta x = 2.5$). The initial data used in this comparison are non-solitary and have a variety of initial phases (D and β are varied in (10)). As in all comparisons with the parent data finite difference, the left panel shows an improvement by a factor of $10\times$ near the training set. In the right panel, the method is better than Fourier collocation at the training data by only a factor of 2. There is, however, a region where the machine-learned difference scheme is $10\times$ better than the Fourier collocation. That this region does not include the training data suggests that data in this region have a particularly large error in Fourier collocation, rather than a particularly small error for the machine-learned difference method.

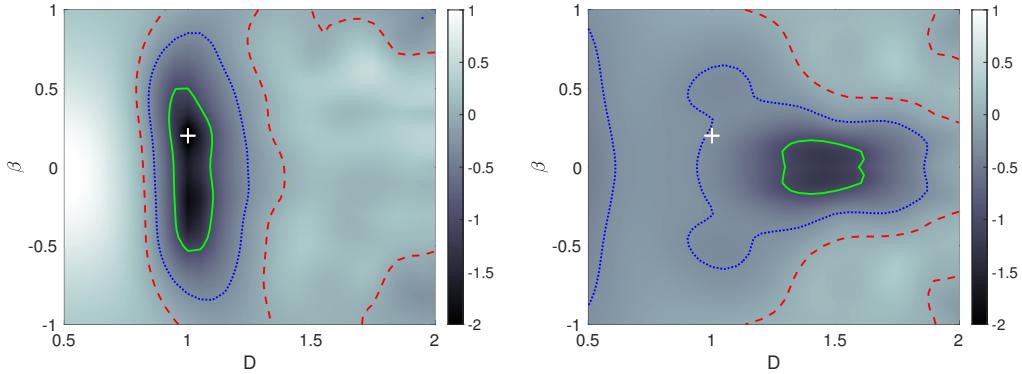

Figure 5. The ratio of the forward error of the machine-learned differencing method to that of its parent finite differencing scheme (**left**) and a Fourier collocation method (**right**) is evaluated for a sampling of initial data, with $\alpha = 0.75$ varying D and β in Equation (10). The solid green contour, marks when the machine-learned solution is $10\times$ more accurate; the dotted blue curve marks where the machine-learned solution is twice as accurate. The methods are equally accurate along the red dashed curve. The location of the training data is marked with a plus sign.

In [37], the three-point discretization of the nonlinearity is observed to avoid chaotic trajectories that the coarse discretization with the single-point discretization is subject to (neither method displays chaotic trajectories if the grid is sufficiently refined). The initial data that were used to observe the chaotic trajectories were designed to trigger the Benjamin–Feir instability; a plane wave and a sideband were used on a small domain with sideband frequency within the Benjamin–Feir instability region, e.g.,

$$u(x,0) = a(1 + \delta\cos(\mu x)).$$

Attempts were made to train the single-point discretization of the nonlinearity on these data. Unfortunately, the optimization procedure did not converge to the prescribed threshold (a tenfold decrease in truncation error). The truncation errors of the parent finite difference scheme are also quite large on these initial data, a natural consequence of the chaotic trajectories. It appears the simple neural network does not have the flexibility required to overcome the size of the errors on this data set. It is possible that a broader and/or deeper network could be used to build a scheme which could suppress the chaos of the parent finite difference scheme in this regime. The multi-point discretization on the other hand, does not have chaotic trajectories in this region, and could be used as a parent with the current network.

4. Conclusions

A machine-learned differencing scheme for the nonlinear Schrödinger equation in the focusing regime is developed. The scheme is designed to have a pure imaginary spectrum, and thus to be linearly stable regardless of training. Two discretizations of the nonlinearity are considered, based on the observed chaos in coarse-gridded finite difference simulations of NLS in [37]. The methods are observed to be $10\times$ more accurate than their parent finite difference schemes for data near the training set. The neural network is embedded into

a symmetric matrix discreteization of the second derivative; thus, when combined with an IMEX time-stepper, it has guaranteed stability. This procedure generalizes to other PDEs with symmetric differential operators, for example, the other Schrödinger equations, the heat equation, or Poisson's equation. The neural network comes with a training cost that traditional methods do not, but can be worth the cost when increased accuracy is necessary.

Author Contributions: Conceptualization, B.F.A.; Software, K.O.F.W.; Investigation, K.O.F.W.; Writing—review & editing, B.F.A.; Writing—review & editing, K.O.F.W.; Funding acquisition, B.F.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Air Force Office of Sponsored research under the program Electromagnetics.

Data Availability Statement: Data is available upon reasonable request.

Acknowledgments: B.F.A. acknowledges funding from the DEJTO and the Air Force Office of Sponsored Research. The authors would also like to thank Jonah Reeger and Arje Nachman for helpful discussions.

Conflicts of Interest: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

1. Williams, K.O.; Akers, B.F. Numerical Simulation of the Korteweg–de Vries Equation with Machine Learning. *Mathematics* **2023**, *11*, 2791. [CrossRef]
2. Pathak, J.; Mustafa, M.; Kashinath, K.; Motheau, E.; Kurth, T.; Day, M. Using machine learning to augment coarse-grid computational fluid dynamics simulations. *arXiv* **2020**, arXiv:2010.00072.
3. Bar-Sinai, Y.; Hoyer, S.; Hickey, J.; Brenner, M.P. Learning data-driven discretizations for partial differential equations. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 15344–15349. [CrossRef]
4. Nordström, J.; Ålund, O. Neural network enhanced computations on coarse grids. *J. Comput. Phys.* **2021**, *425*, 109821. [CrossRef]
5. Fibich, G. *The Nonlinear Schrödinger Equation*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 192.
6. Craik, A.D. *Wave Interactions and Fluid Flows*; Cambridge University Press: Cambridge, UK, 1988.
7. Johnson, R.S. *A Modern Introduction to the Mathematical Theory of Water Waves*; Cambridge University Press: Cambridge, UK, 1997; Number 19.
8. Akers, B.F. Surfactant influence on water wave packets. *Stud. Appl. Math.* **2012**, *129*, 91–102. [CrossRef]
9. Akers, B.F.; Liu, T. Thermal effects in short laser pulses: Suppression of wave collapse. *Wave Motion* **2022**, *115*, 103079. [CrossRef]
10. Zakharov, V.E.; Manakov, S.V. On the complete integrability of a nonlinear Schrödinger equation. *Theor. Math. Phys.* **1974**, *19*, 551–559. [CrossRef]
11. Kath, W.L.; Smyth, N.F. Soliton evolution and radiation loss for the nonlinear Schrödinger equation. *Phys. Rev. E* **1995**, *51*, 1484. [CrossRef]
12. Akers, B.F.; Fiorino, S.T.; Reeger, J.A. Thermal blooming with laser-induced convection: Radial basis function simulation. *Appl. Opt.* **2023**, *62*, G77–G84. [CrossRef]
13. Tolstykh, M.; Frolov, A. Some current problems in numerical weather prediction. *Izv. Atmos. Ocean. Phys.* **2005**, *41*, 285–295.
14. Peregrine, D.H. Water waves, nonlinear Schrödinger equations and their solutions. *ANZIAM J.* **1983**, *25*, 16–43. [CrossRef]
15. Um, K.; Brand, R.; Fei, Y.R.; Holl, P.; Thuerey, N. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 6111–6122.
16. Khodadadian, A.; Parvizi, M.; Teshnehlab, M.; Heitzinger, C. Rational Design of Field-Effect Sensors Using Partial Differential Equations, Bayesian Inversion, and Artificial Neural Networks. *Sensors* **2022**, *22*, 4785. [CrossRef] [PubMed]
17. Noii, N.; Khodadadian, A.; Ulloa, J.; Aldakheel, F.; Wick, T.; Francois, S.; Wriggers, P. Bayesian inversion with open-source codes for various one-dimensional model problems in computational mechanics. *Arch. Comput. Methods Eng.* **2022**, *29*, 4285–4318. [CrossRef]
18. Raissi, M. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *J. Mach. Learn. Res.* **2018**, *19*, 932–955.

19. Long, Z.; Lu, Y.; Ma, X.; Dong, B. Pde-net: Learning pdes from data. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018 ; pp. 3208–3216.
20. Raissi, M.; Karniadakis, G.E. Hidden physics models: Machine learning of nonlinear partial differential equations. *J. Comput. Phys.* **2018**, *357*, 125–141. [CrossRef]
21. Xu, H.; Chang, H.; Zhang, D. DL-PDE: Deep-learning based data-driven discovery of partial differential equations from discrete and noisy data. *arXiv* **2019**, arXiv:1908.04463.
22. Rudy, S.H.; Brunton, S.L.; Proctor, J.L.; Kutz, J.N. Data-driven discovery of partial differential equations. *Sci. Adv.* **2017**, *3*, e1602614. [CrossRef]
23. Wu, K.; Xiu, D. Data-driven deep learning of partial differential equations in modal space. *J. Comput. Phys.* **2020**, *408*, 109307. [CrossRef]
24. Li, J.; Chen, J.; Li, B. Gradient-optimized physics-informed neural networks (GOPINNs): A deep learning method for solving the complex modified KdV equation. *Nonlinear Dyn.* **2022**, *107*, 781–792. [CrossRef]
25. Zhang, Y.; Dong, H.; Sun, J.; Wang, Z.; Fang, Y.; Kong, Y. The new simulation of quasiperiodic wave, periodic wave, and soliton solutions of the KdV-mKdV Equation via a deep learning method. *Comput. Intell. Neurosci.* **2021**, *2021*, 8548482. [CrossRef] [PubMed]
26. Li, J.; Chen, Y. A deep learning method for solving third-order nonlinear evolution equations. *Commun. Theor. Phys.* **2020**, *72*, 115003. [CrossRef]
27. Yang, X.; Wang, Z. Solving Benjamin–Ono equation via gradient balanced PINNs approach. *Eur. Phys. J. Plus* **2022**, *137*, 864. [CrossRef]
28. Bai, Y.; Chaolu, T.; Bilige, S. Physics informed by deep learning: Numerical solutions of modified Korteweg-de Vries equation. *Adv. Math. Phys.* **2021**, *2021*, 5569645. [CrossRef]
29. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
30. Guo, Y.; Cao, X.; Liu, B.; Gao, M. Solving partial differential equations using deep learning and physical constraints. *Appl. Sci.* **2020**, *10*, 5917. [CrossRef]
31. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv* **2017**, arXiv:1711.10561.
32. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning (Adaptive Computation and Machine Learning Series)*; MIT Press: Cambridge, MA, USA, 2017; pp. 321–359.
33. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2019.
34. Ascher, U.M.; Ruuth, S.J.; Spiteri, R.J. Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations. *Appl. Numer. Math.* **1997**, *25*, 151–167.
35. Blechschmidt, J.; Ernst, O.G. Three ways to solve partial differential equations with neural networks—A review. *GAMM-Mitteilungen* **2021**, *44*, e202100006. [CrossRef]
36. Sirignano, J.; Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **2018**, *375*, 1339–1364. [CrossRef]
37. Herbst, B.M.; Ablowitz, M.J. Numerically induced chaos in the nonlinear Schrödinger equation. *Phys. Rev. Lett.* **1989**, *62*, 2065–2068. [CrossRef]